

A Markdown Interpreter for T_EX

Vít Starý Novotný
witiko@mail.muni.cz

Version 3.4.3-0-ge2c6be1a
2024-04-04

Contents

1	Introduction	1	3	Implementation	134
1.1	Requirements	2	3.1	Lua Implementation	134
1.2	Feedback	6	3.2	Plain T _E X Implementation	330
1.3	Acknowledgements	6	3.3	L ^A T _E X Implementation	350
2	Interfaces	6	3.4	ConT _E Xt Implementation	379
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	49			
2.3	L ^A T _E X Interface	124			
2.4	ConT _E Xt Interface	131			
				References	387
				Index	388

List of Figures

1	A block diagram of the Markdown package	7
2	A sequence diagram of typesetting a document using the T _E X interface	45
3	A sequence diagram of typesetting a document using the Lua CLI	46
4	Various formats of mathematical formulae	129
5	The banner of the Markdown package	130

1 Introduction

The Markdown package¹ converts CommonMark² markup to T_EX commands. The functionality is provided both as a Lua module and as plain T_EX, L^AT_EX, and ConT_EXt macro packages that can be used to directly typeset T_EX documents containing markdown markup. Unlike other converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

number of tutorials and code examples. You can find more of these in the user manual.³

```
1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown to plain TeX",
4   author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný",
5   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
6               "2016-2023 Vít Starý Novotný"},
7   license   = "LPPL 1.3c"
8 }
9
10 if not modules then modules = { } end
11 modules['markdown'] = metadata
```

1.1 Requirements

This section gives an overview of all resources required by the package.

1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the Lua_{TEX} engine (though not necessarily in the LuaMeta_{TEX} engine).

LPeg \geq 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg \geq 0.10 is included in Lua_{TEX} \geq 0.72.0 (TeX Live \geq 2013).

```
12 local lpeg = require("lpeg")
```

Selene Unicode A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and note tags to the lower case. Selene Unicode is included in all releases of Lua_{TEX} (TeXLive \geq 2008).

```
13 local unicode = require("unicode")
```

MD5 A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of Lua_{TEX} (TeX Live \geq 2008).

```
14 local md5 = require("md5");
```

Kpathsea A package that implements the loading of third-party Lua libraries and looking up files in the TeX directory structure.

³See <http://mirrors.ctan.org/macros/generic/markdown/markdown.html>.

```
15 (function()
```

If Kpathsea has not been loaded before or if Lua \TeX has not yet been initialized, configure Kpathsea on top of loading it. Since Con \TeX t MkIV provides a `kpse` global that acts as a stub for Kpathsea and the lua-uni-case library expects that `kpse` is a reference to the full Kpathsea library, we load Kpathsea to the `kpse` global.

```
16   local should_initialize = package.loaded.kpse == nil
17                               or tex.initialize ~= nil
18   kpse = require("kpse")
19   if should_initialize then
20     kpse.set_program_name("luatex")
21   end
22 end)()
```

All the abovelisted modules are statically linked into the current version of the Lua \TeX engine [1, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

lua-uni-algos A package that implements Unicode case-folding in \TeX Live \geq 2020.

```
23 local uni_algos = require("lua-uni-algos")
```

api7/lua-tinyyaml A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jeekyllData` option is enabled. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

1.1.2 Plain \TeX Requirements

The plain \TeX part of the package requires that the plain \TeX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.1.1), and the following packages:

expl3 A package that enables the expl3 language from the L \AA \TeX 3 kernel in \TeX Live \leq 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
24 </tex>
25 <*context>
26 \unprotect
27 </context>
28 <*context, tex>
29 \ifx\ExplSyntaxOn\undefined
30   \input expl3-generic
31 \fi
32 </context, tex>
33 <*tex>
```

lt3luabridge A package that allows us to execute Lua code with LuaTeX as well as with other TeX engines that provide the *shell escape* capability, which allows them to execute code with the system’s shell.

The plain TeX part of the package also requires the following Lua module:

Lua File System A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TeX code to create the cache directory specified by the `cacheDir` option before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTeX (TeXLive \geq 2008).

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [1, Section 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [1, Section 4.3].

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.7), the plain TeX part of the package will require that either the LuaTeX `\directlua` primitive or the shell access file stream 18 is available in your TeX engine. If only the shell access file stream is available in your TeX engine (as is the case with pdfTeX and XeTeX), then unless your TeX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.1.3 L^ATeX Requirements

The L^ATeX part of the package requires that the L^ATeX 2_ε format is loaded,

```
34 \NeedsTeXFormat{LaTeX2e}%
```

a TeX engine that extends ϵ -TeX, and all the plain TeX prerequisites (see Section 1.1.2):

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.5) or L^ATeX themes (see Section 2.3.3) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

url A package that provides the `\url` macro for the typesetting of links.

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

paralist A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists as well as the rendering of fancy lists.

- ifthen** A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L^AT_EX theme (see Section 2.3.3).
- fancyvrb** A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code.
- csvsimple** A package that provides the `\csvautotabular` macro for typesetting CSV files in the default renderer prototypes for iA Writer content blocks.
- gobble** A package that provides the `\@gobblethree` T_EX command that is used in the default renderer prototype for citations. The package is included in T_EXLive \geq 2016.
- amsmath and amssymb** Packages that provide symbols used for drawing ticked and unticked boxes.
- catchfile** A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` L^AT_EX theme, see Section 2.3.3.
- graphicx** A package that builds upon the graphics package, which is part of the L^AT_EX 2_ε kernel. It provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.
- grffile** A package that extends the name processing of the graphics package to support a larger range of file names in $2006 \leq \text{T_EX Live} \leq 2019$. Since T_EX Live \geq 2020, the functionality of the package has been integrated in the L^AT_EX 2_ε kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` L^AT_EX themes, see Section 2.3.3.
- etoolbox** A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.5.8, and also in the default renderer prototype for identifier attributes.
- soulutf8** A package that is used in the default renderer prototype for strike-throughs and marked text.
- ltxcmds** A package that is used to detect whether the minted and listings packages are loaded in the default renderer prototype for fenced code blocks.
- verse** A package that is used in the default renderer prototypes for line blocks.

³⁵ `\RequirePackage{expl3}`

1.1.4 ConT_EXt Prerequisites

The ConT_EXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain T_EX prerequisites (see Section 1.1.2), and the following ConT_EXt modules:

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the csv filename extension (see Section 2.2.6).

1.2 Feedback

Please use the Markdown project page on GitHub⁴ to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T_EX-~~L~~A_TE_X Stack Exchange.⁵ community question answering web site under the `markdown` tag.

1.3 Acknowledgements

The Lunamark Lua module provides speedy markdown parsing for the package. I would like to thank John Macfarlane, the creator of Lunamark, for releasing Lunamark under a permissive license, which enabled its use in the Markdown package.

Extensive user documentation for the Markdown package was kindly written by Lian Tze Lim and published by Overleaf.

Funding by the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

Support for content slicing (Lua options `shiftHeadings` and `slice`) and pipe tables (Lua options `pipeTables` and `tableCaptions`) was graciously sponsored by David Vins and Omedym.

The T_EX implementation of the package draws inspiration from several sources including the source code of L^AT_EX 2_ε, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from T_EX, the filecontents package by Scott Pakin and others.

2 Interfaces

This part of the documentation describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package.

Since neither T_EX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of

⁴See <https://github.com/witiko/markdown/issues>.

⁵See <https://tex.stackexchange.com>.

structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to $\text{T}_{\text{E}}\text{X}$ *token renderers* is exposed by the Lua layer. The plain $\text{T}_{\text{E}}\text{X}$ layer exposes the conversion capabilities of Lua as $\text{T}_{\text{E}}\text{X}$ macros. The $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ and $\text{C}^{\text{o}}\text{nT}_{\text{E}}\text{Xt}$ layers provide syntactic sugar on top of plain $\text{T}_{\text{E}}\text{X}$ macros. The user can interface with any and all layers.

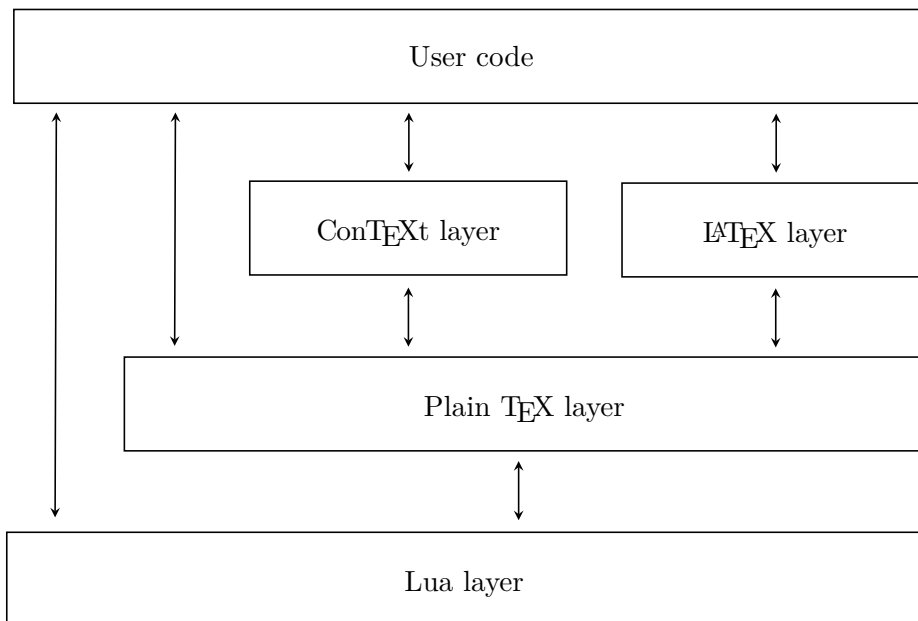


Figure 1: A block diagram of the Markdown package

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain $\text{T}_{\text{E}}\text{X}$. This interface is used by the plain $\text{T}_{\text{E}}\text{X}$ implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

The Lua interface is implemented by the `markdown` Lua module.

```
36 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain $\text{T}_{\text{E}}\text{X}$

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain $\text{T}_{\text{E}}\text{X}$ according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The

`options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

The following example Lua code converts the markdown string `Hello *world*!` to a \TeX output using the default options and prints the \TeX output:

```
local md = require("markdown")
local convert = md.new()
print(convert("Hello *world*!"))
```

2.1.2 User-Defined Syntax Extensions

For the purpose of user-defined syntax extensions, the Lua interface also exposes the `reader` object, which performs the lexical and syntactic analysis of markdown text and which exposes the `reader->insert_pattern` and `reader->add_special_character` methods for extending the PEG grammar of markdown.

The read-only `walkable_syntax` hash table stores those rules of the PEG grammar of markdown that can be represented as an ordered choice of terminal symbols. These rules can be modified by user-defined syntax extensions.

```
37 local walkable_syntax = {
38   Block = {
39     "Blockquote",
40     "Verbatim",
41     "ThematicBreak",
42     "BulletList",
43     "OrderedList",
44     "DisplayHtml",
45     "Heading",
46   },
47   BlockOrParagraph = {
48     "Block",
49     "Paragraph",
50     "Plain",
51   },
52   Inline = {
53     "Str",
54     "Space",
55     "Endline",
56     "EndlineBreak",
57     "LinkAndEmph",
58     "Code",
59     "AutoLinkUrl",
60     "AutoLinkEmail",
61     "AutoLinkRelativeReference",
```



```

62     "InlineHtml",
63     "HtmlEntity",
64     "EscapedChar",
65     "Smart",
66     "Symbol",
67   },
68 }

```

The `reader->insert_pattern` method inserts a PEG pattern into the grammar of markdown. The method receives two mandatory arguments: a selector string in the form "*<left-hand side terminal symbol> <before, after, or instead of> <right-hand side terminal symbol>*" and a PEG pattern to insert, and an optional third argument with a name of the PEG pattern for debugging purposes (see the `debugExtensions` option). The name does not need to be unique and shall not be interpreted by the Markdown package; you can treat it as a comment.

For example. if we'd like to insert `pattern` into the grammar between the `Inline -> LinkAndEmph` and `Inline -> Code` rules, we would call `reader->insert_pattern` with `"Inline after LinkAndEmph"` (or `"Inline before Code"`) and `pattern` as the arguments.

The `reader->add_special_character` method adds a new character with special meaning to the grammar of markdown. The method receives the character as its only argument.

2.1.3 Options

The Lua interface recognizes the following options. When unspecified, the value of a key is taken from the `defaultOptions` table.

```

69 local defaultOptions = {}

```

To enable the enumeration of Lua options, we will maintain the `\g_@@_lua_options_seq` sequence.

```

70 \ExplSyntaxOn
71 \seq_new:N \g_@@_lua_options_seq

```

To enable the reflection of default Lua options and their types, we will maintain the `\g_@@_default_lua_options_prop` and `\g_@@_lua_option_types_prop` property lists, respectively.

```

72 \prop_new:N \g_@@_lua_option_types_prop
73 \prop_new:N \g_@@_default_lua_options_prop
74 \seq_new:N \g_@@_option_layers_seq
75 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
76 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
77 \cs_new:Nn
78   \@@_add_lua_option:nnn
79   {
80     \@@_add_option:Vnnn

```

```

81     \c_@@_option_layer_lua_tl
82     { #1 }
83     { #2 }
84     { #3 }
85   }
86 \cs_new:Nn
87   \@@_add_option:nmmn
88   {
89     \seq_gput_right:cn
90     { g_@@_ #1 _options_seq }
91     { #2 }
92     \prop_gput:cnn
93     { g_@@_ #1 _option_types_prop }
94     { #2 }
95     { #3 }
96     \prop_gput:cnn
97     { g_@@_default_ #1 _options_prop }
98     { #2 }
99     { #4 }
100   \@@_typecheck_option:n
101     { #2 }
102   }
103 \cs_generate_variant:Nn
104   \@@_add_option:nmmn
105   { Vmmn }
106 \tl_const:Nn \c_@@_option_value_true_tl { true }
107 \tl_const:Nn \c_@@_option_value_false_tl { false }
108 \cs_new:Nn \@@_typecheck_option:n
109   {
110     \@@_get_option_type:nN
111     { #1 }
112     \l_tmpa_tl
113     \str_case_e:Vn
114     \l_tmpa_tl
115     {
116       { \c_@@_option_type_boolean_tl }
117       {
118         \@@_get_option_value:nN
119         { #1 }
120         \l_tmpa_tl
121         \bool_if:nF
122         {
123           \str_if_eq_p:VV
124           \l_tmpa_tl
125           \c_@@_option_value_true_tl ||
126           \str_if_eq_p:VV
127           \l_tmpa_tl

```

```

128         \c_@@_option_value_false_tl
129     }
130     {
131         \msg_error:nnnV
132         { markdown }
133         { failed-typecheck-for-boolean-option }
134         { #1 }
135         \l_tmpa_tl
136     }
137 }
138 }
139 }
140 \msg_new:nnn
141 { markdown }
142 { failed-typecheck-for-boolean-option }
143 {
144     Option~#1~has~value~#2,~
145     but~a~boolean~(true~or~false)~was~expected.
146 }
147 \cs_generate_variant:Nn
148   \str_case_e:nn
149   { Vn }
150 \cs_generate_variant:Nn
151   \msg_error:nnnn
152   { nnnV }
153 \seq_new:N \g_@@_option_types_seq
154 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
155 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
156 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
157 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
158 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
159 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
160 \tl_const:Nn \c_@@_option_type_number_tl { number }
161 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
162 \tl_const:Nn \c_@@_option_type_path_tl { path }
163 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
164 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
165 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
166 \tl_const:Nn \c_@@_option_type_string_tl { string }
167 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
168 \cs_new:Nn
169   \@@_get_option_type:nN
170   {
171     \bool_set_false:N
172       \l_tmpa_bool
173     \seq_map_inline:Nn
174       \g_@@_option_layers_seq

```

```

175     {
176     \prop_get:cnNT
177     { g_@@_ ##1 _option_types_prop }
178     { #1 }
179     \l_tmpa_tl
180     {
181     \bool_set_true:N
182     \l_tmpa_bool
183     \seq_map_break:
184     }
185     }
186 \bool_if:nF
187 \l_tmpa_bool
188 {
189 \msg_error:nnn
190 { markdown }
191 { undefined-option }
192 { #1 }
193 }
194 \seq_if_in:NVF
195 \g_@@_option_types_seq
196 \l_tmpa_tl
197 {
198 \msg_error:nnnV
199 { markdown }
200 { unknown-option-type }
201 { #1 }
202 \l_tmpa_tl
203 }
204 \tl_set_eq:NN
205 #2
206 \l_tmpa_tl
207 }
208 \msg_new:nnn
209 { markdown }
210 { unknown-option-type }
211 {
212 Option~#1~has~unknown~type~#2.
213 }
214 \msg_new:nnn
215 { markdown }
216 { undefined-option }
217 {
218 Option~#1~is~undefined.
219 }
220 \cs_new:Nn
221 \@@_get_default_option_value:nN

```

```

222 {
223   \bool_set_false:N
224     \l_tmpa_bool
225   \seq_map_inline:Nn
226     \g_@@_option_layers_seq
227     {
228       \prop_get:cnNT
229         { g_@@_default_ ##1 _options_prop }
230         { #1 }
231         #2
232         {
233           \bool_set_true:N
234             \l_tmpa_bool
235           \seq_map_break:
236         }
237       }
238   \bool_if:nF
239     \l_tmpa_bool
240     {
241       \msg_error:nnn
242         { markdown }
243         { undefined-option }
244         { #1 }
245     }
246 }
247 \cs_new:Nn
248   \@@_get_option_value:nN
249   {
250     \@@_option_tl_to_csname:nN
251     { #1 }
252     \l_tmpa_tl
253   \cs_if_free:cTF
254     { \l_tmpa_tl }
255     {
256       \@@_get_default_option_value:nN
257       { #1 }
258       #2
259     }
260   {
261     \@@_get_option_type:nN
262     { #1 }
263     \l_tmpa_tl
264   \str_if_eq:NNTF
265     \c_@@_option_type_counter_tl
266     \l_tmpa_tl
267     {
268       \@@_option_tl_to_csname:nN

```

```

269         { #1 }
270         \l_tmpa_tl
271         \tl_set:Nx
272         #2
273         { \the \cs:w \l_tmpa_tl \cs_end: }
274     }
275     {
276         \@@_option_tl_to_csname:nN
277         { #1 }
278         \l_tmpa_tl
279         \tl_set:Nv
280         #2
281         { \l_tmpa_tl }
282     }
283 }
284 }
285 \cs_new:Nn \@@_option_tl_to_csname:nN
286 {
287     \tl_set:Nn
288     \l_tmpa_tl
289     { \str_uppercase:n { #1 } }
290     \tl_set:Nx
291     #2
292     {
293         markdownOption
294         \tl_head:f { \l_tmpa_tl }
295         \tl_tail:n { #1 }
296     }
297 }

```

To make it easier to support different coding styles in the interface, engines, we define the `\@@_with_various_cases:nn` function that allows us to generate different variants of a string using different cases.

```

298 \cs_new:Nn \@@_with_various_cases:nn
299 {
300     \seq_clear:N
301     \l_tmpa_seq
302     \seq_map_inline:Nn
303     \g_@@_cases_seq
304     {
305         \tl_set:Nn
306         \l_tmpa_tl
307         { #1 }
308         \use:c { ##1 }
309         \l_tmpa_tl
310         \seq_put_right:NV
311         \l_tmpa_seq

```

```

312         \l_tmpa_tl
313     }
314     \seq_map_inline:Nn
315         \l_tmpa_seq
316         { #2 }
317 }

```

To interrupt the `\@@_with_various_cases:n` function prematurely, use the `\@@_with_various_cases_break:` function.

```

318 \cs_new:Nn \@@_with_various_cases_break:
319 {
320     \seq_map_break:
321 }

```

By default, `camelCase` and `snake_case` are supported. Additional cases can be added by adding functions to the `\g_@@_cases_seq` sequence.

```

322 \seq_new:N \g_@@_cases_seq
323 \cs_new:Nn \@@_camel_case:N
324 {
325     \regex_replace_all:mnN
326         { _ ([a-z]) }
327         { \c { str_uppercase:n } \cB\{ \1 \cE\} }
328         #1
329     \tl_set:Nx
330         #1
331         { #1 }
332 }
333 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
334 \cs_new:Nn \@@_snake_case:N
335 {
336     \regex_replace_all:mnN
337         { ([a-z])([A-Z]) }
338         { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
339         #1
340     \tl_set:Nx
341         #1
342         { #1 }
343 }
344 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 General Behavior

`eagerCache=true, false`

default: `false`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also

produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

This behavior will always be used if the `finalizeCache` option is enabled.

false Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

This behavior will only be used when the `finalizeCache` option is disabled.

```
345 \@@_add_lua_option:nnn
346   { eagerCache }
347   { boolean }
348   { false }

349 defaultOptions.eagerCache = false
```

`singletonCache=true, false`

default: true

true Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time- and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions. This has been the default behavior since version 3.0.0 of the Markdown package.

false Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also issue #226⁶.

This was the default behavior until version 3.0.0 of the Markdown package.

```
350 \@@_add_lua_option:nnn
351   { singletonCache }
352   { boolean }
353   { true }

354 defaultOptions.singletonCache = true

355 local singletonCache = {
356   convert = nil,
357   options = nil,
358 }
```

⁶See <https://github.com/witiko/markdown/pull/226#issuecomment-1599641634>.

2.1.5 File and Directory Names

`cacheDir`= $\langle path \rangle$ default: .

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain T_EX implementations. The Lua implementation expects that the entire path already exists.

When iteratively writing and typesetting a markdown document, the cache files are going to accumulate over time. You are advised to clean the cache directory every now and then, or to set it to a temporary filesystem (such as `/tmp` on UN*X systems), which gets periodically emptied.

```
359 \@@_add_lua_option:nnn
360   { cacheDir }
361   { path }
362   { \markdownOptionOutputDir / _markdown_\jobname }
363 defaultOptions.cacheDir = "."
```

`contentBlocksLanguageMap`= $\langle filename \rangle$
default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.5.9 for more information.

```
364 \@@_add_lua_option:nnn
365   { contentBlocksLanguageMap }
366   { path }
367   { markdown-languages.json }
368 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`debugExtensionsFileName`= $\langle filename \rangle$ default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```
369 \@@_add_lua_option:nnn
370   { debugExtensionsFileName }
371   { path }
372   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
373 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

`frozenCacheFileName`= $\langle path \rangle$ default: `frozenCache.tex`

A path to an output file (frozen cache) that will be created when the `finalizeCache` option is enabled and will contain a mapping between an enumeration of markdown documents and their auxiliary cache files.

The frozen cache makes it possible to later typeset a plain \TeX document that contains markdown documents without invoking Lua using the `frozenCache` plain \TeX option. As a result, the plain \TeX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
374 \@@_add_lua_option:nnn
375   { frozenCacheFileName }
376   { path }
377   { \markdownOptionCacheDir / frozenCache.tex }
378 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

2.1.6 Parser Options

`autoIdentifiers`=true, false default: false

true Enable the Pandoc auto identifiers syntax extension⁷:

The following heading received the identifier ``sesame-street``:

```
# 123 Sesame Street
```

false Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```
379 \@@_add_lua_option:nnn
380   { autoIdentifiers }
381   { boolean }
382   { false }
383 defaultOptions.autoIdentifiers = false
```

`blankBeforeBlockquote`=true, false default: false

true Require a blank line between a paragraph and the following blockquote.

false Do not require a blank line between a paragraph and the following blockquote.

⁷See https://pandoc.org/MANUAL.html#extension-auto_identifiers.

```
384 \@@_add_lua_option:nnn
385   { blankBeforeBlockquote }
386   { boolean }
387   { false }

388 defaultOptions.blankBeforeBlockquote = false
```

`blankBeforeCodeFence=true, false` default: false

- true** Require a blank line between a paragraph and the following fenced code block.
- false** Do not require a blank line between a paragraph and the following fenced code block.

```
389 \@@_add_lua_option:nnn
390   { blankBeforeCodeFence }
391   { boolean }
392   { false }

393 defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeDivFence=true, false` default: false

- true** Require a blank line before the closing fence of a fenced div.
- false** Do not require a blank line before the closing fence of a fenced div.

```
394 \@@_add_lua_option:nnn
395   { blankBeforeDivFence }
396   { boolean }
397   { false }

398 defaultOptions.blankBeforeDivFence = false
```

`blankBeforeHeading=true, false` default: false

- true** Require a blank line between a paragraph and the following header.
- false** Do not require a blank line between a paragraph and the following header.

```
399 \@@_add_lua_option:nnn
400   { blankBeforeHeading }
401   { boolean }
402   { false }

403 defaultOptions.blankBeforeHeading = false
```

`blankBeforeList=true, false` default: false

- `true` Require a blank line between a paragraph and the following list.
- `false` Do not require a blank line between a paragraph and the following list.

```
404 \@@_add_lua_option:nnn
405   { blankBeforeList }
406   { boolean }
407   { false }

408 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

- `true` Enable the Pandoc bracketed span syntax extension⁸:

`[This is *some text*]{.class key=val}`

- `false` Disable the Pandoc bracketed span syntax extension.

```
409 \@@_add_lua_option:nnn
410   { bracketedSpans }
411   { boolean }
412   { false }

413 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: true

- `true` A blank line separates block quotes.
- `false` Blank lines in the middle of a block quote are ignored.

```
414 \@@_add_lua_option:nnn
415   { breakableBlockquotes }
416   { boolean }
417   { true }

418 defaultOptions.breakableBlockquotes = true
```

⁸See https://pandoc.org/MANUAL.html#extension-bracketed_spans.

`citationNbsps=true, false`

default: `false`

`true` Replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

`false` Do not replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

```
419 \@@_add_lua_option:nnn
420 { citationNbsps }
421 { boolean }
422 { true }

423 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: `false`

`true` Enable the Pandoc citation syntax extension⁹:

Here is a simple parenthetical citation [`@doe99`] and here is a string of several [`see @doe99, pp. 33-35; also @smith04, chap. 1`].

A parenthetical citation can have a [`prenote @doe99`] and a [`@smith04 postnote`]. The name of the author can be suppressed by inserting a dash before the name of an author as follows [`-@smith04`].

Here is a simple text citation `@doe99` and here is a string of several `@doe99` [`pp. 33-35; also @smith04, chap. 1`]. Here is one with the name of the author suppressed `-@doe99`.

`false` Disable the Pandoc citation syntax extension.

```
424 \@@_add_lua_option:nnn
425 { citations }
426 { boolean }
427 { false }

428 defaultOptions.citations = false
```

⁹See <https://pandoc.org/MANUAL.html#extension-citations>.

`codeSpans=true, false`

default: true

true Enable the code span syntax:

```
Use the printf() function.  
``There is a literal backtick (`) here.``
```

false Disable the code span syntax. This allows you to easily use the quotation mark ligatures in texts that do not contain code spans:

```
``This is a quote.``
```

```
429 \@@_add_lua_option:nnn  
430 { codeSpans }  
431 { boolean }  
432 { true }  
  
433 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

true

: Enable the iA Writer content blocks syntax extension [3]:

```
``` md  
http://example.com/minard.jpg (Napoleon's
 disastrous Russian campaign of 1812)
/Flowchart.png "Engineering Flowchart"
/Savings Account.csv 'Recent Transactions'
/Example.swift
/Lorem Ipsum.txt
.....
```

**false** Disable the iA Writer content blocks syntax extension.

```
434 \@@_add_lua_option:nnn
435 { contentBlocks }
436 { boolean }
437 { false }

438 defaultOptions.contentBlocks = false
```

`contentLevel=block, inline` default: block

**block** Treat content as a sequence of blocks.

```
- this is a list
- it contains two items
```

**inline** Treat all content as inline content.

```
- this is a text
- not a list
```

```
439 \@@_add_lua_option:nnn
440 { contentLevel }
441 { string }
442 { block }
443 defaultOptions.contentLevel = "block"
```

`debugExtensions=true, false` default: false

**true** Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

**false** Do not produce a JSON file with the PEG grammar of markdown.

```
444 \@@_add_lua_option:nnn
445 { debugExtensions }
446 { boolean }
447 { false }
448 defaultOptions.debugExtensions = false
```

`definitionLists=true, false` default: false

**true** Enable the pandoc definition list syntax extension:

```
Term 1

: Definition 1

Term 2 with inline markup
```

```

: Definition 2

 { some code, part of Definition 2 }

Third paragraph of definition 2.

```

**false**      Disable the pandoc definition list syntax extension.

```

449 \@@_add_lua_option:nnn
450 { definitionLists }
451 { boolean }
452 { false }

453 defaultOptions.definitionLists = false

```

**expectJekyllData=true, false**

default: false

**false**      When the `jekyllData` option is enabled, then a markdown document may begin with YAML metadata if and only if the metadata begin with the end-of-directives marker (`---`) and they end with either the end-of-directives or the end-of-document marker (`...`):

```

\documentclass{article}
\usepackage[jekyllData]{markdown}
\begin{document}
\begin{markdown}

- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- Markdown
\end{markdown}
\end{document}

```



`true` When the `jeekyllData` option is enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

```
\documentclass{article}
\usepackage[jekyllData, expectJekyllData]{markdown}
\begin{document}
\begin{markdown}
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}
```

```
454 \@@_add_lua_option:nnn
455 { expectJekyllData }
456 { boolean }
457 { false }
458 defaultOptions.expectJekyllData = false
```

`extensions=<filenames>`

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the `kpathsea` library is available, files will be searched for not only in the current working directory but also in the  $\TeX$  directory structure.

A user-defined syntax extension is a Lua file in the following format:

```
local strike_through = {
 api_version = 2,
 grammar_version = 4,
 finalize_grammar = function(reader)
 local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
 local doubleslashes = lpeg.P("//")
```

```

local function between(p, starter, ender)
 ender = lpeg.B(nonspacechar) * ender
 return (starter * #nonspacechar
 * lpeg.Ct(p * (p - ender)^0) * ender)
end

local read_strike_through = between(
 lpeg.V("Inline"), doubleslashes, doubleslashes
) / function(s) return {"\\st{" , s, "}"} end

reader.insert_pattern("Inline after LinkAndEmph", read_strike_through,
 "StrikeThrough")
reader.add_special_character("/")
end
}

return strike_through

```

The `api_version` and `grammar_version` fields specify the version of the user-defined syntax extension API and the markdown grammar for which the extension was written. See the current API and grammar versions below:

```

459 metadata.user_extension_api_version = 2
460 metadata.grammar_version = 4

```

Any changes to the syntax extension API or grammar will cause the corresponding current version to be incremented. After Markdown 3.0.0, any changes to the API and the grammar will be either backwards-compatible or constitute a breaking change that will cause the major version of the Markdown package to increment (to 4.0.0).

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `reader` object, such as the `reader->insert_pattern` and `reader->add_special_character` methods, see Section 2.1.2.

```

461 \cs_generate_variant:Nn
462 \@@_add_lua_option:nnn
463 { nnV }
464 \@@_add_lua_option:nnV
465 { extensions }
466 { clist }
467 \c_empty_clist
468 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

**true** Enable the Pandoc fancy list syntax extension<sup>10</sup>:

```
a) first item
b) second item
c) third item
```

**false** Disable the Pandoc fancy list syntax extension.

```
469 \@@_add_lua_option:nnn
470 { fancyLists }
471 { boolean }
472 { false }
473 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: true

**true** Enable the commonmark fenced code block extension:

```
~~~ js
if (a > 3) {
    moveShip(5 * gravity, DOWN);
}
~~~~~

``` html
<pre>
  <code>
    // Some comments
    line 1 of code
    line 2 of code
    line 3 of code
  </code>
</pre>
```
```

**false** Disable the commonmark fenced code block extension.

```
474 \@@_add_lua_option:nnn
475 { fencedCode }
476 { boolean }
477 { true }
478 defaultOptions.fencedCode = true
```

<sup>10</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

`fencedCodeAttributes=true, false`

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>11</sup>:

```
~~~~ {#mycode .haskell .numberLines startFrom=100}
qsort []      = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
                qsort (filter (>= x) xs)
~~~~~
```

**false** Disable the Pandoc fenced code attribute syntax extension.

```
479 \@@_add_lua_option:nnn
480 { fencedCodeAttributes }
481 { boolean }
482 { false }

483 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

**true** Enable the Pandoc fenced div syntax extension<sup>12</sup>:

```
::::: {#special .sidebar}
Here is a paragraph.

And another.
:::::
```

**false** Disable the Pandoc fenced div syntax extension.

```
484 \@@_add_lua_option:nnn
485 { fencedDivs }
486 { boolean }
487 { false }

488 defaultOptions.fencedDivs = false
```

<sup>11</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-fenced_code_attributes).

<sup>12</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

`finalizeCache=true, false`

default: `false`

Whether an output file specified with the `frozenCacheFileName` option (frozen cache) that contains a mapping between an enumeration of markdown documents and their auxiliary cache files will be created.

The frozen cache makes it possible to later typeset a plain  $\text{T}_{\text{E}}\text{X}$  document that contains markdown documents without invoking Lua using the `frozenCache` plain  $\text{T}_{\text{E}}\text{X}$  option. As a result, the plain  $\text{T}_{\text{E}}\text{X}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
489 \@@_add_lua_option:nnn
490 { finalizeCache }
491 { boolean }
492 { false }

493 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>`

default: `0`

The number of the current markdown document that will be stored in an output file (frozen cache) when the `finalizeCache` is enabled. When the document number is 0, then a new frozen cache will be created. Otherwise, the frozen cache will be appended.

Each frozen cache entry will define a  $\text{T}_{\text{E}}\text{X}$  macro `\markdownFrozenCache<number>` that will typeset markdown document number `<number>`.

```
494 \@@_add_lua_option:nnn
495 { frozenCacheCounter }
496 { counter }
497 { 0 }

498 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: `false`

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>13</sup>:

```
The following heading received the identifier `123-sesame-street`:

123 Sesame Street
```

`false` Disable the Pandoc GitHub-flavored auto identifiers syntax extension.

---

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

See also the option [autoIdentifiers](#).

```
499 \@@_add_lua_option:nnn
500 { gfmAutoIdentifiers }
501 { boolean }
502 { false }

503 defaultOptions.gfmAutoIdentifiers = false
```

`hashEnumerators=true, false`

default: `false`

`true` Enable the use of hash symbols (#) as ordered item list markers:

```
#. Bird
#. McHale
#. Parish
```

`false` Disable the use of hash symbols (#) as ordered item list markers.

```
504 \@@_add_lua_option:nnn
505 { hashEnumerators }
506 { boolean }
507 { false }

508 defaultOptions.hashEnumerators = false
```

`headerAttributes=true, false`

default: `false`

`true` Enable the assignment of HTML attributes to headings:

```
My first heading {#foo}

My second heading ## {#bar .baz}

Yet another heading {key=value}
=====
```

`false` Disable the assignment of HTML attributes to headings.

```
509 \@@_add_lua_option:nnn
510 { headerAttributes }
511 { boolean }
512 { false }

513 defaultOptions.headerAttributes = false
```

`html=true, false` default: true

- true** Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.
- false** Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

```
514 \@@_add_lua_option:nnn
515 { html }
516 { boolean }
517 { true }

518 defaultOptions.html = true
```

`hybrid=true, false` default: false

- true** Disable the escaping of special plain  $\TeX$  characters, which makes it possible to intersperse your markdown markup with  $\TeX$  code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix  $\TeX$  and markdown markup freely.
- false** Enable the escaping of special plain  $\TeX$  characters outside verbatim environments, so that they are not interpreted by  $\TeX$ . This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```
519 \@@_add_lua_option:nnn
520 { hybrid }
521 { boolean }
522 { false }

523 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: false

- true** Enable the Pandoc inline code span attribute extension<sup>14</sup>:

``<$>`{.haskell}`

---

<sup>14</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-inline_code_attributes).

`false` Enable the Pandoc inline code span attribute extension.

```
524 \@@_add_lua_option:nnn
525 { inlineCodeAttributes }
526 { boolean }
527 { false }

528 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

`true` Enable the Pandoc inline note syntax extension<sup>15</sup>:

```
Here is an inline note.^[Inlines notes are easier to
write, since you don't have to pick an identifier and
move down to type the note.]
```

`false` Disable the Pandoc inline note syntax extension.

```
529 \@@_add_lua_option:nnn
530 { inlineNotes }
531 { boolean }
532 { false }

533 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>16</sup> for entering metadata in YAML:

```

title: 'This is the title: it contains a colon'
author:
- Author One
- Author Two
keywords: [nothing, nothingness]
abstract: |
 This is the abstract.

 It consists of two paragraphs.

```

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).



**false** Disable the Pandoc YAML metadata block syntax extension for entering metadata in YAML.

```
534 \@@_add_lua_option:nnn
535 { jekyllData }
536 { boolean }
537 { false }
538 defaultOptions.jekyllData = false
```

**linkAttributes=true, false** default: false

**true** Enable the Pandoc link and image attribute syntax extension<sup>17</sup>:

An inline `![image](foo.jpg){#id .class width=30 height=20px}` and a reference `![image][ref]` with attributes.

`[ref]: foo.jpg "optional title" {#id .class key=val key2=val2}`

**false** Enable the Pandoc link and image attribute syntax extension.

```
539 \@@_add_lua_option:nnn
540 { linkAttributes }
541 { boolean }
542 { false }
543 defaultOptions.linkAttributes = false
```

**lineBlocks=true, false** default: false

**true** Enable the Pandoc line block syntax extension<sup>18</sup>:

```
| this is a line block that
| spans multiple
| even
| discontinuous
| lines
```

**false** Disable the Pandoc line block syntax extension.

```
544 \@@_add_lua_option:nnn
545 { lineBlocks }
546 { boolean }
547 { false }
548 defaultOptions.lineBlocks = false
```

---

<sup>17</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

`mark=true, false` default: false

`true` Enable the Pandoc mark syntax extension<sup>19</sup>:

```
This ==is highlighted text.==
```

`false` Disable the Pandoc mark syntax extension.

```
549 \@@_add_lua_option:nnn
550 { mark }
551 { boolean }
552 { false }
553 defaultOptions.mark = false
```

`notes=true, false` default: false

`true` Enable the Pandoc note syntax extension<sup>20</sup>:

```
Here is a note reference, [^1] and another. [^longnote]
```

```
[^1]: Here is the note.
```

```
[^longnote]: Here's one with multiple blocks.
```

```
 Subsequent paragraphs are indented to show that they
 belong to the previous note.
```

```
 { some.code }
```

```
 The whole paragraph can be indented, or just the
 first line. In this way, multi-paragraph notes
 work like multi-paragraph list items.
```

```
This paragraph won't be part of the note, because it
isn't indented.
```

`false` Disable the Pandoc note syntax extension.

```
554 \@@_add_lua_option:nnn
555 { notes }
556 { boolean }
557 { false }
558 defaultOptions.notes = false
```

<sup>19</sup>See <https://pandoc.org/MANUAL.html#extension-mark>.

<sup>20</sup>See <https://pandoc.org/MANUAL.html#extension-footnotes>.

`pipeTables=true, false`

default: false

**true** Enable the PHP Markdown pipe table syntax extension:

| Right | Left | Default | Center |
|-------|------|---------|--------|
| 12    | 12   | 12      | 12     |
| 123   | 123  | 123     | 123    |
| 1     | 1    | 1       | 1      |

**false** Disable the PHP Markdown pipe table syntax extension.

```
559 \@@_add_lua_option:nnn
560 { pipeTables }
561 { boolean }
562 { false }

563 defaultOptions.pipeTables = false
```

`preserveTabs=true, false`

default: true

**true** Preserve tabs in code block and fenced code blocks.

**false** Convert any tabs in the input to spaces.

```
564 \@@_add_lua_option:nnn
565 { preserveTabs }
566 { boolean }
567 { true }

568 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>21</sup>:

```
`$H_2 O$`{=tex} is a liquid.
```

To enable raw blocks, the `fencedCode` option must also be enabled:

```
Here is a mathematical formula:
```{=tex}
\[distance[i] =
  \begin{dcases}
    a & b \\
  \end{dcases}
\]
```

²¹See https://pandoc.org/MANUAL.html#extension-raw_attribute.

```
      c & d
    \end{dcases}
\]
---
```

The `rawAttribute` option is a good alternative to the `hybrid` option. Unlike the `hybrid` option, which affects the entire document, the `rawAttribute` option allows you to isolate the parts of your documents that use TeX:

`false` Disable the Pandoc raw attribute syntax extension.

```
569 \@@_add_lua_option:nnn
570   { rawAttribute }
571   { boolean }
572   { false }

573 defaultOptions.rawAttribute = false
```

`relativeReferences=true, false`

default: `false`

`true` Enable relative references²² in autolinks:

```
I conclude in Section <#conclusion>.

Conclusion {#conclusion}
=====

In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
```

`false` Disable relative references in autolinks.

```
574 \@@_add_lua_option:nnn
575   { relativeReferences }
576   { boolean }
577   { false }

578 defaultOptions.relativeReferences = false
```

²²See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

`shiftHeadings`=*<shift amount>*

default: 0

All headings will be shifted by *<shift amount>*, which can be both positive and negative. Headings will not be shifted beyond level 6 or below level 1. Instead, those headings will be shifted to level 6, when *<shift amount>* is positive, and to level 1, when *<shift amount>* is negative.

```
579 \@@_add_lua_option:nnn
580   { shiftHeadings }
581   { number }
582   { 0 }

583 defaultOptions.shiftHeadings = 0
```

`slice`=*<the beginning and the end of a slice>*

default: `^ $`

Two space-separated selectors that specify the slice of a document that will be processed, whereas the remainder of the document will be ignored. The following selectors are recognized:

- The circumflex (`^`) selects the beginning of a document.
- The dollar sign (`$`) selects the end of a document.
- `^<identifier>` selects the beginning of a section (see the `headerAttributes` option) or a fenced div (see the `fencedDivs` option) with the HTML attribute `#<identifier>`.
- `$<identifier>` selects the end of a section with the HTML attribute `#<identifier>`.
- `<identifier>` corresponds to `^<identifier>` for the first selector and to `$<identifier>` for the second selector.

Specifying only a single selector, `<identifier>`, is equivalent to specifying the two selectors `<identifier> <identifier>`, which is equivalent to `^<identifier> $<identifier>`, i.e. the entire section with the HTML attribute `#<identifier>` will be selected.

```
584 \@@_add_lua_option:nnn
585   { slice }
586   { slice }
587   { ^-$ }

588 defaultOptions.slice = "^ $"
```

`smartEllipses=true, false` default: false

`true` Convert any ellipses in the input to the `\markdownRendererEllipsis` \TeX macro.

`false` Preserve all ellipses in the input.

```
589 \@@_add_lua_option:nnn
590 { smartEllipses }
591 { boolean }
592 { false }

593 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

`true` Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererOListItemWithNumber` \TeX macro.

`false` Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOListItem` \TeX macro.

```
594 \@@_add_lua_option:nnn
595 { startNumber }
596 { boolean }
597 { true }

598 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension²³:

`This is deleted text.`

`false` Disable the Pandoc strike-through syntax extension.

```
599 \@@_add_lua_option:nnn
600 { strikeThrough }
601 { boolean }
602 { false }

603 defaultOptions.strikeThrough = false
```

²³See <https://pandoc.org/MANUAL.html#extension-strikeout>.

`stripIndent=true, false`

default: `false`

`true` Strip the minimal indentation of non-blank lines from all lines in a markdown document. Requires that the `preserveTabs` Lua option is disabled:

```
\documentclass{article}
\usepackage[stripIndent]{markdown}
\begin{document}
  \begin{markdown}
    Hello *world*!
  \end{markdown}
\end{document}
```

`false` Do not strip any indentation from the lines in a markdown document.

```
604 \@@_add_lua_option:nnn
605   { stripIndent }
606   { boolean }
607   { false }
608 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

`true` Enable the Pandoc subscript syntax extension²⁴:

```
H~2~0 is a liquid.
```

`false` Disable the Pandoc subscript syntax extension.

```
609 \@@_add_lua_option:nnn
610   { subscripts }
611   { boolean }
612   { false }
613 defaultOptions.subscripts = false
```

²⁴See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`superscripts=true, false`

default: `false`

`true` Enable the Pandoc superscript syntax extension²⁵:

```
2^10^ is 1024.
```

`false` Disable the Pandoc superscript syntax extension.

```
614 \@@_add_lua_option:nnn
615   { superscripts }
616   { boolean }
617   { false }

618 defaultOptions.superscripts = false
```

`tableAttributes=true, false`

default: `false`

`true`

: Enable the assignment of HTML attributes to table captions (see the `tableCaptions` option).

```
``` md
| Right | Left | Default | Center |
|-----:|:-----|-----:|:-----:|
| 12 | 12 | 12 | 12 |
| 123 | 123 | 123 | 123 |
| 1 | 1 | 1 | 1 |

: Demonstration of pipe table syntax. {#example-table}
```
```

`false` Disable the assignment of HTML attributes to table captions.

```
619 \@@_add_lua_option:nnn
620   { tableAttributes }
621   { boolean }
622   { false }

623 defaultOptions.tableAttributes = false
```

²⁵See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension²⁶ for pipe tables (see the `pipeTables` option).

```
``` md
| Right | Left | Default | Center |
|-----:|:-----|-----:|:-----:|
| 12 | 12 | 12 | 12 |
| 123 | 123 | 123 | 123 |
| 1 | 1 | 1 | 1 |

: Demonstration of pipe table syntax.
.....
```

`false` Disable the Pandoc table caption syntax extension.

```
624 \@@_add_lua_option:nnn
625 { tableCaptions }
626 { boolean }
627 { false }

628 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension<sup>27</sup>:

```
- [] an unticked task list item
- [/] a half-checked task list item
- [X] a ticked task list item
```

`false` Disable the Pandoc task list syntax extension.

```
629 \@@_add_lua_option:nnn
630 { taskLists }
631 { boolean }
632 { false }

633 defaultOptions.taskLists = false
```

<sup>26</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>27</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

`texComments=true, false`

default: false

**true** Strip T<sub>E</sub>X-style comments.

```
\documentclass{article}
\usepackage[texComments]{markdown}
\begin{document}
\begin{markdown}
Hello *world*!
\end{markdown}
\end{document}
```

Always enabled when `hybrid` is enabled.

**false** Do not strip T<sub>E</sub>X-style comments.

```
634 \@@_add_lua_option:nnn
635 { texComments }
636 { boolean }
637 { false }
638 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

**true** Enable the Pandoc dollar math syntax extension<sup>28</sup>:

```
inline math: $E=mc^2$
display math: $$E=mc^2$$
```

**false** Disable the Pandoc dollar math syntax extension.

```
639 \@@_add_lua_option:nnn
640 { texMathDollars }
641 { boolean }
642 { false }
643 defaultOptions.texMathDollars = false
```

---

<sup>28</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_dollars](https://pandoc.org/MANUAL.html#extension-tex_math_dollars).

`texMathDoubleBackslash=true, false` default: false

**true** Enable the Pandoc double backslash math syntax extension<sup>29</sup>:

```
inline math: \\(E=mc^2\\)
display math: \\[E=mc^2\\]
```

**false** Disable the Pandoc double backslash math syntax extension.

```
644 \\@@_add_lua_option:nnn
645 { texMathDoubleBackslash }
646 { boolean }
647 { false }

648 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

**true** Enable the Pandoc single backslash math syntax extension<sup>30</sup>:

```
inline math: \ (E=mc^2\)
display math: \ [E=mc^2\]
```

**false** Disable the Pandoc single backslash math syntax extension.

```
649 \\@@_add_lua_option:nnn
650 { texMathSingleBackslash }
651 { boolean }
652 { false }

653 defaultOptions.texMathSingleBackslash = false
```

`tightLists=true, false` default: true

**true** Unordered and ordered lists whose items do not consist of multiple paragraphs will be considered *tight*. Tight lists will produce tight renderers that may produce different output than lists that are not tight:

---

<sup>29</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>30</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_single\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash).

```

- This is
- a tight
- unordered list.

- This is

 not a tight

- unordered list.

```

**false** Unordered and ordered lists whose items consist of multiple paragraphs will be treated the same way as lists that consist of multiple paragraphs.

```

654 \@@_add_lua_option:nmn
655 { tightLists }
656 { boolean }
657 { true }

658 defaultOptions.tightLists = true

```

**underscores=true, false**

default: true

**true** Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```

single asterisks
single underscores
double asterisks
__double underscores__

```

**false** Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the **hybrid** option without the need to constantly escape subscripts.

```

659 \@@_add_lua_option:nmn
660 { underscores }
661 { boolean }
662 { true }
663 \ExplSyntaxOff

664 defaultOptions.underscores = true

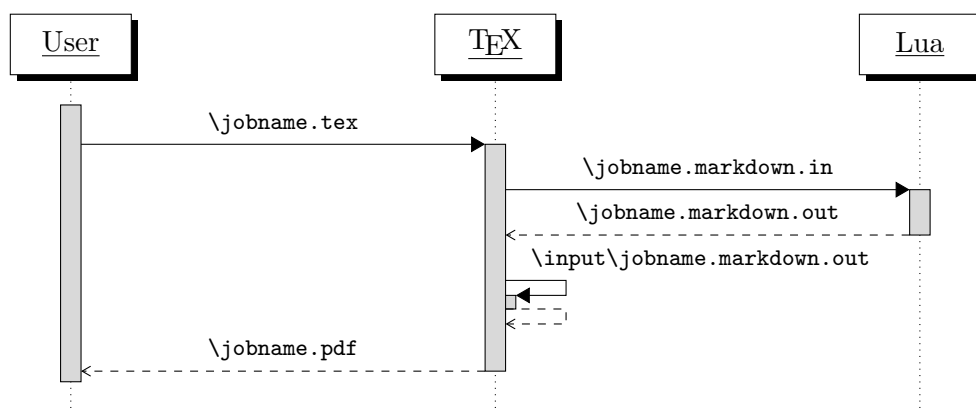
```

### 2.1.7 Command-Line Interface

The high-level operation of the Markdown package involves the communication between several programming layers: the plain  $\text{T}_{\text{E}}\text{X}$  layer hands markdown documents to the Lua layer. Lua converts the documents to  $\text{T}_{\text{E}}\text{X}$ , and hands the converted documents back to plain  $\text{T}_{\text{E}}\text{X}$  layer for typesetting, see Figure 2.

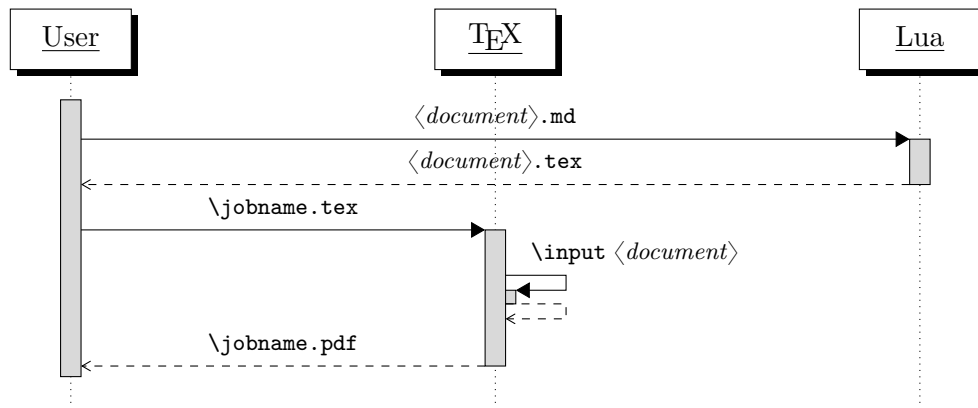
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{T}_{\text{E}}\text{X}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{T}_{\text{E}}\text{X}$  engine includes a Lua interpreter, the package also requires shell access, which opens the door for a malicious actor to access the system. Last, but not least, the complexity of the procedure impedes debugging.

A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to  $\text{T}_{\text{E}}\text{X}$  is also provided, see Figure 3.



**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the  $\text{T}_{\text{E}}\text{X}$  interface**

```
665
666 local HELP_STRING = [[
667 Usage: texlua]] .. arg[0] .. [[[OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
668 where OPTIONS are documented in the Lua interface section of the
669 technical Markdown package documentation.
670
671 When OUTPUT_FILE is unspecified, the result of the conversion will be
672 written to the standard output. When INPUT_FILE is also unspecified, the
673 result of the conversion will be read from the standard input.
674
675 Report bugs to: witiko@mail.muni.cz
676 Markdown package home page: <https://github.com/witiko/markdown>]]
677
```



**Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface**

```

678 local VERSION_STRING = [[
679 markdown-cli.lua (Markdown)]] .. metadata.version .. [[
680
681 Copyright (C)]] .. table.concat(metadata.copyright,
682 "\nCopyright (C) ") .. [[
683
684 License:]] .. metadata.license
685
686 local function warn(s)
687 io.stderr:write("Warning: " .. s .. "\n") end
688
689 local function error(s)
690 io.stderr:write("Error: " .. s .. "\n")
691 os.exit(1)
692 end

```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```

693 local function camel_case(option_name)
694 local cased_option_name = option_name:gsub("_(%l)", function(match)
695 return match:sub(2, 2):upper()
696 end)
697 return cased_option_name
698 end
699
700 local function snake_case(option_name)
701 local cased_option_name = option_name:gsub("%l%u", function(match)
702 return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()

```

```

703 end)
704 return cased_option_name
705 end
706
707 local cases = {camel_case, snake_case}
708 local various_case_options = {}
709 for option_name, _ in pairs(defaultOptions) do
710 for _, case in ipairs(cases) do
711 various_case_options[case(option_name)] = option_name
712 end
713 end
714
715 local process_options = true
716 local options = {}
717 local input_filename
718 local output_filename
719 for i = 1, #arg do
720 if process_options then

```

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

```

721 if arg[i] == "--" then
722 process_options = false
723 goto continue

```

Unless the `--` argument has been specified before, an argument containing the equals sign (=) is assumed to be an option specification in a `<key>=<value>` format. The available options are listed in Section 2.1.3.

```

724 elseif arg[i]:match("=") then
725 local key, value = arg[i]:match("(.)=(.*)")
726 if defaultOptions[key] == nil and
727 various_case_options[key] ~= nil then
728 key = various_case_options[key]
729 end

```

The `defaultOptions` table is consulted to identify whether `<value>` should be parsed as a string, number, table, or boolean.

```

730 local default_type = type(defaultOptions[key])
731 if default_type == "boolean" then
732 options[key] = (value == "true")
733 elseif default_type == "number" then
734 options[key] = tonumber(value)
735 elseif default_type == "table" then
736 options[key] = {}
737 for item in value:gmatch("[^,]+") do
738 table.insert(options[key], item)

```

```

739 end
740 else
741 if default_type ~= "string" then
742 if default_type == "nil" then
743 warn('Option "' .. key .. '" not recognized.')
744 else
745 warn('Option "' .. key .. '" type not recognized, please file ' ..
746 'a report to the package maintainer.')
747 end
748 warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
749 key .. '" as a string.')
750 end
751 options[key] = value
752 end
753 goto continue

```

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

```

754 elseif arg[i] == "--help" or arg[i] == "-h" then
755 print(HELP_STRING)
756 os.exit()

```

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

```

757 elseif arg[i] == "--version" or arg[i] == "-v" then
758 print(VERSION_STRING)
759 os.exit()
760 end
761 end

```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a  $\text{\TeX}$  document.

```

762 if input_filename == nil then
763 input_filename = arg[i]

```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the  $\text{\TeX}$  document that will result from the conversion.

```

764 elseif output_filename == nil then
765 output_filename = arg[i]
766 else
767 error('Unexpected argument: "' .. arg[i] .. "'.')
768 end
769 ::continue::
770 end

```



The command-line Lua interface is implemented by the `markdown-cli.lua` file that can be invoked from the command line as follows:

```
texlua /path/to/markdown-cli.lua cacheDir=. -- hello.md hello.tex
```

to convert the Markdown document `hello.md` to a TeX document `hello.tex`. After the Markdown package for our TeX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

## 2.2 Plain TeX Interface

The plain TeX interface provides macros for the typesetting of markdown input from within plain TeX, for setting the Lua interface options (see Section 2.1.3) used during the conversion from markdown to plain TeX and for changing the way markdown the tokens are rendered.

```
771 \def\markdownLastModified{((LASTMODIFIED))}%
772 \def\markdownVersion{((VERSION))}%
```

The plain TeX interface is implemented by the `markdown.tex` file that can be loaded as follows:

```
\input markdown
```

It is expected that the special plain TeX characters have the expected category codes, when `\inputting` the file.

### 2.2.1 Typesetting Markdown

The interface exposes the `\markdownBegin`, `\markdownEnd`, `\markdownInput`, and `\markdownEscape` macros.

The `\markdownBegin` macro marks the beginning of a markdown document fragment and the `\markdownEnd` macro marks its end.

```
773 \let\markdownBegin\relax
774 \let\markdownEnd\relax
```

You may prepend your own code to the `\markdownBegin` macro and redefine the `\markdownEnd` macro to produce special effects before and after the markdown block.

There are several limitations to the macros you need to be aware of. The first limitation concerns the `\markdownEnd` macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corollary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [6, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

The `\markdownBegin` and `\markdownEnd` macros will also consume the rest of the lines at which they appear. In the following example plain T<sub>E</sub>X code, the characters `c`, `e`, and `f` will not appear in the output.

```
\input markdown
a
b \markdownBegin c
d
e \markdownEnd f
g
\bye
```

Note that you may also not nest the `\markdownBegin` and `\markdownEnd` macros.

The following example plain T<sub>E</sub>X code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

```
\input markdown
\markdownBegin
Hello **world** ...
\markdownEnd
\bye
```

You can use the `\markdownInput` macro to include markdown documents, similarly to how you might use the `\input` T<sub>E</sub>X primitive to include T<sub>E</sub>X documents. The `\markdownInput` macro accepts a single parameter with the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X.

```
775 \let\markdownInput\relax
```

This macro is not subject to the abovelisted limitations of the `\markdownBegin` and `\markdownEnd` macros.

The following example plain T<sub>E</sub>X code showcases the usage of the `\markdownInput` macro:

```
\input markdown
\markdownInput{hello.md}
\bye
```

The `\markdownEscape` macro accepts a single parameter with the filename of a  $\TeX$  document and executes the  $\TeX$  document in the middle of a markdown document fragment. Unlike the `\input` built-in of  $\TeX$ , `\markdownEscape` guarantees that the standard catcode regime of your  $\TeX$  format will be used.

```
776 \let\markdownEscape\relax
```

## 2.2.2 Options

The plain  $\TeX$  options are represented by  $\TeX$  commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain  $\TeX$  interface.

To determine whether plain  $\TeX$  is the top layer or if there are other layers above plain  $\TeX$ , we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that plain  $\TeX$  is the top layer.

```
777 \ExplSyntaxOn
778 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
779 \cs_generate_variant:Nn
780 \tl_const:Nn
781 { NV }
782 \tl_if_exist:NF
783 \c_@@_top_layer_tl
784 {
785 \tl_const:NV
786 \c_@@_top_layer_tl
787 \c_@@_option_layer_plain_tex_tl
788 }
```

To enable the enumeration of plain  $\TeX$  options, we will maintain the `\g_@@_plain_tex_options_seq` sequence.

```
789 \seq_new:N \g_@@_plain_tex_options_seq
```

To enable the reflection of default plain  $\TeX$  options and their types, we will maintain the `\g_@@_default_plain_tex_options_prop` and `\g_@@_plain_tex_option_types_prop` property lists, respectively.

```
790 \prop_new:N \g_@@_plain_tex_option_types_prop
791 \prop_new:N \g_@@_default_plain_tex_options_prop
792 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
793 \cs_new:Nn
794 \@@_add_plain_tex_option:nnn
795 {
796 \@@_add_option:Vnnn
797 \c_@@_option_layer_plain_tex_tl
798 { #1 }
799 { #2 }
800 { #3 }
801 }
```

The plain T<sub>E</sub>X options may be also be specified via the `\markdownSetup` macro. Here, the plain T<sub>E</sub>X options are represented by a comma-delimited list of `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` if the `=<value>` part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```

802 \cs_new:Nn
803 \@@_setup:n
804 {
805 \keys_set:nn
806 { markdown/options }
807 { #1 }
808 }
809 \cs_gset_eq:NN
810 \markdownSetup
811 \@@_setup:n

```

The `\markdownIfOption{<name>}{<iftrue>}{<iffalse>}` macro is provided for testing, whether the value of `\markdownOption<name>` is `true`. If the value is `true`, then `<iftrue>` is expanded, otherwise `<iffalse>` is expanded.

```

812 \prg_new_conditional:Nnn
813 \@@_if_option:n
814 { TF, T, F }
815 {
816 \@@_get_option_type:nN
817 { #1 }
818 \l_tmpa_tl
819 \str_if_eq:NNF
820 \l_tmpa_tl
821 \c_@@_option_type_boolean_tl
822 {
823 \msg_error:nxxx
824 { markdown }
825 { expected-boolean-option }
826 { #1 }
827 { \l_tmpa_tl }
828 }
829 \@@_get_option_value:nN
830 { #1 }
831 \l_tmpa_tl
832 \str_if_eq:NNTF
833 \l_tmpa_tl
834 \c_@@_option_value_true_tl
835 { \prg_return_true: }
836 { \prg_return_false: }
837 }
838 \msg_new:nnn
839 { markdown }

```

```

840 { expected-boolean-option }
841 {
842 Option~#1~has~type~#2,~
843 but~a~boolean~was~expected.
844 }
845 \let\markdownIfOption=\@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `frozenCacheFileName` (frozen cache) that contains a mapping between an enumeration of the markdown documents in the plain T<sub>E</sub>X document and their auxiliary files cached in the `cacheDir` directory.

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `finalizeCache` option, and uses it to typeset the plain T<sub>E</sub>X document without invoking Lua. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

```

846 \@@_add_plain_tex_option:nnn
847 { frozenCache }
848 { boolean }
849 { false }

```

The standard usage of the above two options is as follows:

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `finalizeCache` option.
4. Typeset the plain T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X document and the `cacheDir` directory.

**2.2.2.2 File and Directory Names** The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`.

The expansion of this macro must not contain quotation marks (") or backslash symbols (\). Mind that T<sub>E</sub>X engines tend to put quotation marks around `\jobname`, when it contains spaces.

```

850 \@@_add_plain_tex_option:nnn
851 { inputTempFileName }
852 { path }
853 { \jobname.markdown.in }

```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain T<sub>E</sub>X implementation. The option defaults to `.`

or, since TeX Live 2024, to the value of the `-output-directory` option of your TeX engine.

The path must be set to the same value as the `-output-directory` option of your TeX engine for the package to function correctly. We need this macro to make the Lua implementation aware where it should store the helper files. The same limitations apply here as in the case of the `inputTempFileName` macro.

The `\markdownOptionOutputDir` macro has been deprecated and will be removed in the next major version of the Markdown package.

```
854 \cs_generate_variant:Nn
855 \@@_add_plain_tex_option:nnn
856 { nnV }
```

Use the `lt3luabridge` library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since TeX Live 2024.

```
857 \ExplSyntaxOff
858 \input lt3luabridge.tex
859 \ExplSyntaxOn
860 \bool_if:nTF
861 {
862 \cs_if_exist_p:N
863 \luabridge_tl_set:Nn &&
864 (
865 \int_compare_p:nNn
866 { \g_luabridge_method_int }
867 =
868 { \c_luabridge_method_directlua_int } ||
869 \sys_if_shell_unrestricted_p:
870)
871 }
872 {
873 \luabridge_tl_set:Nn
874 \l_tmpa_tl
875 { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
876 }
877 {
878 \tl_set:Nn
879 \l_tmpa_tl
880 { . }
881 }
882 \@@_add_plain_tex_option:nnV
883 { outputDir }
884 { path }
885 \l_tmpa_tl
```

### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section [sec:#themes](#)). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

The `\markdownOptionPlain` macro specifies whether higher-level T<sub>E</sub>X formats should only use the plain T<sub>E</sub>X default definitions or whether they should also use the format-specific default definitions. Whereas plain T<sub>E</sub>X default definitions only provide definitions for simple elements such as emphasis, strong emphasis, and paragraph separators, format-specific default definitions add support for more complex elements such as lists, tables, and citations. On the flip side, plain T<sub>E</sub>X default definitions load no extra resources and are rather stable, whereas format-specific default definitions load extra resources and are subject to a more rapid change.

Here is how you would enable the macro in a L<sup>A</sup>T<sub>E</sub>X document:

```
\usepackage[plain]{markdown}
```

Here is how you would enable the macro in a ConT<sub>E</sub>Xt document:

```
\def\markdownOptionPlain{true}
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
886 \@_add_plain_tex_option:nnn
887 { plain }
888 { boolean }
889 { false }
```

The `\markdownOptionNoDefaults` macro specifies whether we should prevent the loading of default definitions or not. This is useful in contexts, where we want to have total control over how all elements are rendered.

Here is how you would enable the macro in a L<sup>A</sup>T<sub>E</sub>X document:

```
\usepackage[noDefaults]{markdown}
```

Here is how you would enable the macro in a ConT<sub>E</sub>Xt document:

```
\def\markdownOptionNoDefaults{true}
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
890 \@@_add_plain_tex_option:nnn
891 { noDefaults }
892 { boolean }
893 { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.5) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
894 \seq_gput_right:Nn
895 \g_@@_plain_tex_options_seq
896 { stripPercentSigns }
897 \prop_gput:Nnn
898 \g_@@_plain_tex_option_types_prop
899 { stripPercentSigns }
900 { boolean }
901 \prop_gput:Nnx
902 \g_@@_default_plain_tex_options_prop
903 { stripPercentSigns }
904 { false }
```

#### 2.2.2.5 Generating Plain T<sub>E</sub>X Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain T<sub>E</sub>X macros and the key-value interface of the `\markdownSetup` macro for the above plain T<sub>E</sub>X options.

The command also defines macros and key-values that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain T<sub>E</sub>X implementation, only passed along to Lua.

Furthermore, the command also defines options and key-values for subsequently loaded layers that correspond to higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

For the macros that correspond to the non-boolean options recognized by the Lua interface, the same limitations apply here in the case of the `inputTempFileName` macro.

```
905 \cs_new:Nn
906 \@@_define_option_commands_and_keyvals:
907 {
908 \seq_map_inline:Nn
909 \g_@@_option_layers_seq
```



```

910 {
911 \seq_map_inline:cn
912 { g_@@_ ##1 _options_seq }
913 {
914 \@@_define_option_command:n
915 { #####1 }

```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```

916 \@@_with_various_cases:nm
917 { #####1 }
918 {
919 \@@_define_option_keyval:nmm
920 { ##1 }
921 { #####1 }
922 { #####1 }
923 }
924 }
925 }
926 }
927 \cs_new:Nn
928 \@@_define_option_command:n
929 {

```

Do not override options defined before loading the package.

```

930 \@@_option_tl_to_csname:nN
931 { #1 }
932 \l_tmpa_tl
933 \cs_if_exist:cF
934 { \l_tmpa_tl }
935 {
936 \@@_get_default_option_value:nN
937 { #1 }
938 \l_tmpa_tl
939 \@@_set_option_value:nV
940 { #1 }
941 \l_tmpa_tl
942 }
943 }
944 \cs_new:Nn
945 \@@_set_option_value:nn
946 {
947 \@@_define_option:n
948 { #1 }
949 \@@_get_option_type:nN
950 { #1 }

```

```

951 \l_tmpa_tl
952 \str_if_eq:NNTF
953 \c_@@_option_type_counter_tl
954 \l_tmpa_tl
955 {
956 \@@_option_tl_to_csname:nN
957 { #1 }
958 \l_tmpa_tl
959 \int_gset:cn
960 { \l_tmpa_tl }
961 { #2 }
962 }
963 {
964 \@@_option_tl_to_csname:nN
965 { #1 }
966 \l_tmpa_tl
967 \cs_set:cpn
968 { \l_tmpa_tl }
969 { #2 }
970 }
971 }
972 \cs_generate_variant:Nn
973 \@@_set_option_value:nn
974 { nV }
975 \cs_new:Nn
976 \@@_define_option:n
977 {
978 \@@_option_tl_to_csname:nN
979 { #1 }
980 \l_tmpa_tl
981 \cs_if_free:cT
982 { \l_tmpa_tl }
983 {
984 \@@_get_option_type:nN
985 { #1 }
986 \l_tmpb_tl
987 \str_if_eq:NNT
988 \c_@@_option_type_counter_tl
989 \l_tmpb_tl
990 {
991 \@@_option_tl_to_csname:nN
992 { #1 }
993 \l_tmpa_tl
994 \int_new:c
995 { \l_tmpa_tl }
996 }
997 }

```

```

998 }
999 \cs_new:Nn
1000 \@@_define_option_keyval:nnn
1001 {
1002 \prop_get:cnN
1003 { g_@@_ #1 _option_types_prop }
1004 { #2 }
1005 \l_tmpa_tl
1006 \str_if_eq:VVTF
1007 \l_tmpa_tl
1008 \c_@@_option_type_boolean_tl
1009 {
1010 \keys_define:nn
1011 { markdown/options }
1012 {

```

For boolean options, we also accept **yes** as an alias for **true** and **no** as an alias for **false**.

```

1013 #3 .code:n = {
1014 \tl_set:Nx
1015 \l_tmpa_tl
1016 {
1017 \str_case:nnF
1018 { ##1 }
1019 {
1020 { yes } { true }
1021 { no } { false }
1022 }
1023 { ##1 }
1024 }
1025 \@@_set_option_value:nV
1026 { #2 }
1027 \l_tmpa_tl
1028 },
1029 #3 .default:n = { true },
1030 }
1031 }
1032 {
1033 \keys_define:nn
1034 { markdown/options }
1035 {
1036 #3 .code:n = {
1037 \@@_set_option_value:nn
1038 { #2 }
1039 { ##1 }
1040 },
1041 }

```

```
1042 }
```

For options of type `clist`, we assume that  $\langle key \rangle$  is a regular English noun in plural (such as `extensions`) and we also define the  $\langle singular\ key \rangle = \langle value \rangle$  interface, where  $\langle singular\ key \rangle$  is  $\langle key \rangle$  after stripping the trailing `-s` (such as `extension`). Rather than setting the option to  $\langle value \rangle$ , this interface appends  $\langle value \rangle$  to the current value as the rightmost item in the list.

```
1043 \str_if_eq:VVT
1044 \l_tmpa_tl
1045 \c_@@_option_type_clist_tl
1046 {
1047 \tl_set:Nn
1048 \l_tmpa_tl
1049 { #3 }
1050 \tl_reverse:N
1051 \l_tmpa_tl
1052 \str_if_eq:enF
1053 {
1054 \tl_head:V
1055 \l_tmpa_tl
1056 }
1057 { s }
1058 {
1059 \msg_error:nnn
1060 { markdown }
1061 { malformed-name-for-clist-option }
1062 { #3 }
1063 }
1064 \tl_set:Nx
1065 \l_tmpa_tl
1066 {
1067 \tl_tail:V
1068 \l_tmpa_tl
1069 }
1070 \tl_reverse:N
1071 \l_tmpa_tl
1072 \tl_put_right:Nn
1073 \l_tmpa_tl
1074 {
1075 .code:n = {
1076 \@@_get_option_value:nN
1077 { #2 }
1078 \l_tmpa_tl
1079 \clist_set:NV
1080 \l_tmpa_clist
1081 { \l_tmpa_tl, { ##1 } }
1082 \@@_set_option_value:nV
```

```

1083 { #2 }
1084 \l_tmpa_clist
1085 }
1086 }
1087 \keys_define:nV
1088 { markdown/options }
1089 \l_tmpa_tl
1090 }
1091 }
1092 \cs_generate_variant:Nn
1093 \clist_set:Nn
1094 { NV }
1095 \cs_generate_variant:Nn
1096 \keys_define:nn
1097 { nV }
1098 \cs_generate_variant:Nn
1099 \@@_set_option_value:nn
1100 { nV }
1101 \prg_generate_conditional_variant:Nnn
1102 \str_if_eq:nn
1103 { en }
1104 { F }
1105 \msg_new:nnn
1106 { markdown }
1107 { malformed-name-for-clist-option }
1108 {
1109 Clist-option-name~#1~does~not~end~with~-s.
1110 }

```

If plain T<sub>E</sub>X is the top layer, we use the `\@@_define_option_commands_and_keyvals:` macro to define plain T<sub>E</sub>X option macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

1111 \str_if_eq:VVT
1112 \c_@@_top_layer_tl
1113 \c_@@_option_layer_plain_tex_tl
1114 {
1115 \@@_define_option_commands_and_keyvals:
1116 }
1117 \ExplSyntaxOff

```

### 2.2.3 Themes

User-defined themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The key-values `theme=<theme name>` and `import=<theme name>` load a T<sub>E</sub>X document (further referred to as *a theme*) named `markdowntheme<munged theme`

*name*⟩.tex, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name is *qualified* and contains no underscores. A theme name is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is  $\langle theme\ author\rangle/\langle theme\ purpose\rangle/\langle private\ naming\ scheme\rangle$ , where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the T<sub>E</sub>X directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T<sub>E</sub>X document package named `markdownthemewitiko_beamer_MU.tex`.

```

1118 \ExplSyntaxOn
1119 \keys_define:nn
1120 { markdown/options }
1121 {
1122 theme .code:n = {
1123 \@@_set_theme:n
1124 { #1 }
1125 },
1126 import .code:n = {
1127 \tl_set:Nn
1128 \l_tmpa_tl
1129 { #1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1130 \tl_replace_all:NnV
1131 \l_tmpa_tl
1132 { / }
1133 \c_backslash_str
1134 \keys_set:nV
1135 { markdown/options/import }
1136 \l_tmpa_tl
1137 },
1138 }

```

To keep track of the current theme when themes are nested, we will maintain the `\g_@@_themes_seq` stack of theme names. For convenience, the name of the current theme is also available in the `\g_@@_current_theme_tl` macro.

```

1139 \seq_new:N
1140 \g_@@_themes_seq
1141 \tl_new:N
1142 \g_@@_current_theme_tl
1143 \tl_gset:Nn
1144 \g_@@_current_theme_tl
1145 { }
1146 \seq_gput_right:NV
1147 \g_@@_themes_seq
1148 \g_@@_current_theme_tl
1149 \cs_new:Nn
1150 \@@_set_theme:n
1151 {

```

First, we validate the theme name.

```

1152 \str_if_in:nnF
1153 { #1 }
1154 { / }
1155 {
1156 \msg_error:nnn
1157 { markdown }
1158 { unqualified-theme-name }
1159 { #1 }
1160 }
1161 \str_if_in:nnT
1162 { #1 }
1163 { _ }
1164 {
1165 \msg_error:nnn
1166 { markdown }
1167 { underscores-in-theme-name }
1168 { #1 }
1169 }

```

Next, we munge the theme name.

```

1170 \str_set:Nn
1171 \l_tmpa_str
1172 { #1 }
1173 \str_replace_all:Nnn
1174 \l_tmpa_str
1175 { / }
1176 { _ }

```

Finally, we load the theme.

```

1177 \tl_gset:Nn
1178 \g_@@_current_theme_tl
1179 { #1 / }
1180 \seq_gput_right:NV

```

```

1181 \g_@@_themes_seq
1182 \g_@@_current_theme_tl
1183 \@@_load_theme:nV
1184 { #1 }
1185 \l_tmpa_str
1186 \seq_gpop_right:NN
1187 \g_@@_themes_seq
1188 \l_tmpa_tl
1189 \seq_get_right:NN
1190 \g_@@_themes_seq
1191 \l_tmpa_tl
1192 \tl_gset:NV
1193 \g_@@_current_theme_tl
1194 \l_tmpa_tl
1195 }
1196 \msg_new:nnnn
1197 { markdown }
1198 { unqualified-theme-name }
1199 { Won't load theme with unqualified name~#1 }
1200 { Theme names must contain at least one forward slash }
1201 \msg_new:nnnn
1202 { markdown }
1203 { underscores-in-theme-name }
1204 { Won't load theme with an underscore in its name~#1 }
1205 { Theme names must not contain underscores in their names }
1206 \cs_generate_variant:Nn
1207 \tl_replace_all:Nnn
1208 { NnV }
1209 \ExplSyntaxOff

```

Built-in plain T<sub>E</sub>X themes provided with the Markdown package include:

**witiko/tilde** A theme that makes tilde (~) always typeset the non-breaking space even when the **hybrid** Lua option is disabled.

```


\input markdown
\markdownSetup{import=witiko/tilde}
\markdownBegin
Bartel~Leendert van~der~Waerden
\markdownEnd
\bye

```

Typesetting the above document produces the following text: “Bartel Leendert van der Waerden”.



**witiko/markdown/defaults** A plain T<sub>E</sub>X theme with the default definitions of token renderer prototypes for plain T<sub>E</sub>X. This theme is loaded automatically together with the package and explicitly loading it has no effect.

Please, see Section 3.2.2 for implementation details of the built-in plain T<sub>E</sub>X themes.

## 2.2.4 Snippets

We may set up options as *snippets* using the `\markdownSetupSnippet` macro and invoke them later. The `\markdownSetupSnippet` macro receives two arguments: the name of the snippet and the options to store.

```
1210 \ExplSyntaxOn
1211 \prop_new:N
1212 \g_@@_snippets_prop
1213 \cs_new:Nn
1214 \@@_setup_snippet:nn
1215 {
1216 \tl_if_empty:nT
1217 { #1 }
1218 {
1219 \msg_error:nnn
1220 { markdown }
1221 { empty-snippet-name }
1222 { #1 }
1223 }
1224 \tl_set:NV
1225 \l_tmpa_tl
1226 \g_@@_current_theme_tl
1227 \tl_put_right:Nn
1228 \l_tmpa_tl
1229 { #1 }
1230 \@@_if_snippet_exists:nT
1231 { #1 }
1232 {
1233 \msg_warning:nnV
1234 { markdown }
1235 { redefined-snippet }
1236 \l_tmpa_tl
1237 }
1238 \prop_gput:NVn
1239 \g_@@_snippets_prop
1240 \l_tmpa_tl
1241 { #2 }
1242 }
1243 \cs_gset_eq:NN
```

```

1244 \markdownSetupSnippet
1245 \@@_setup_snippet:nn
1246 \msg_new:nnnn
1247 { markdown }
1248 { empty-snippet-name }
1249 { Empty-snippet-name~#1 }
1250 { Pick-a-non-empty-name-for-your-snippet }
1251 \msg_new:nnn
1252 { markdown }
1253 { redefined-snippet }
1254 { Redefined~snippet~#1 }

```

To decide whether a snippet exists, we can use the `\markdownIfSnippetExists` macro.

```

1255 \prg_new_conditional:Nnn
1256 \@@_if_snippet_exists:n
1257 { TF, T, F }
1258 {
1259 \tl_set:NV
1260 \l_tmpa_tl
1261 \g_@@_current_theme_tl
1262 \tl_put_right:Nn
1263 \l_tmpa_tl
1264 { #1 }
1265 \prop_get:NVNTF
1266 \g_@@_snippets_prop
1267 \l_tmpa_tl
1268 \l_tmpb_tl
1269 { \prg_return_true: }
1270 { \prg_return_false: }
1271 }
1272 \cs_gset_eq:NN
1273 \markdownIfSnippetExists
1274 \@@_if_snippet_exists:nTF

```

The option with key `snippet` invokes a snippet named  $\langle value \rangle$ .

```

1275 \keys_define:nn
1276 { markdown/options }
1277 {
1278 snippet .code:n = {
1279 \tl_set:NV
1280 \l_tmpa_tl
1281 \g_@@_current_theme_tl
1282 \tl_put_right:Nn
1283 \l_tmpa_tl
1284 { #1 }
1285 \@@_if_snippet_exists:nTF
1286 { #1 }

```

```

1287 {
1288 \prop_get:NVN
1289 \g_@@_snippets_prop
1290 \l_tmpa_tl
1291 \l_tmpb_tl
1292 \@@_setup:V
1293 \l_tmpb_tl
1294 }
1295 {
1296 \msg_error:nnV
1297 { markdown }
1298 { undefined-snippet }
1299 \l_tmpa_tl
1300 }
1301 }
1302 }
1303 \msg_new:nnn
1304 { markdown }
1305 { undefined-snippet }
1306 { Can't invoke undefined snippet~#1 }
1307 \cs_generate_variant:Nn
1308 \@@_setup:n
1309 { V }
1310 \ExplSyntaxOff

```

Here is how we can use snippets to store options and invoke them later in L<sup>A</sup>T<sub>E</sub>X:

```

\markdownSetupSnippet{romanNumerals}{
 renderers = {
 olItemWithNumber = {\item[\romannumeral#1\relax.]},
 },
}
\begin{markdown}

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

\end{markdown}
\begin{markdown}[snippet=romanNumerals]

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

If the `romanNumerals` snippet were defined in the `jdooe/lists` theme, we could import the `jdooe/lists` theme and use the qualified name `jdooe/lists/romanNumerals` to invoke the snippet:

```
\markdownSetup{import=jdooe/lists}
\begin{markdown}[snippet=jdooe/lists/romanNumerals]

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

Alternatively, we can use the extended variant of the `import` L<sup>A</sup>T<sub>E</sub>X option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

```
\markdownSetup{
 import = {
 jdooe/lists = romanNumerals,
 },
}
\begin{markdown}[snippet=romanNumerals]

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

Furthermore, we can also specify the name of the snippet in the current namespace, which can be different from the name of the snippet in the `jdooe/lists` theme. For example, we can make the snippet `jdooe/lists/romanNumerals` available under the name `roman`.

```
\markdownSetup{
 import = {
```

```

 jdoe/lists = romanNumerals as roman,
 },
}
\begin{markdown}[snippet=roman]

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

Several themes and/or snippets can be loaded at once using the extended variant of the `import` L<sup>A</sup>T<sub>E</sub>X option:

```

\markdownSetup{
 import = {
 jdoe/longpackagename/lists = {
 arabic as arabic1,
 roman,
 alphabetic,
 },
 jdoe/anotherlongpackagename/lists = {
 arabic as arabic2,
 },
 jdoe/yetanotherlongpackagename,
 },
}

```

```

1311 \ExplSyntaxOn
1312 \tl_new:N
1313 \l_@@_import_current_theme_tl
1314 \keys_define:nn
1315 { markdown/options/import }
1316 {

```

If a theme name is given without a list of snippets to import, we assume that an empty list was given.

```

1317 unknown .default:n = {},
1318 unknown .code:n = {

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and

then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1319 \tl_set_eq:NN
1320 \l_@@_import_current_theme_tl
1321 \l_keys_key_str
1322 \tl_replace_all:NVN
1323 \l_@@_import_current_theme_tl
1324 \c_backslash_str
1325 { / }

```

Here, we import the snippets.

```

1326 \clist_map_inline:nn
1327 { #1 }
1328 {
1329 \regex_extract_once:nnNTF
1330 { ^(.*)\s+as\s+(.*)$ }
1331 { ##1 }
1332 \l_tmpa_seq
1333 {
1334 \seq_pop:NN
1335 \l_tmpa_seq
1336 \l_tmpa_tl
1337 \seq_pop:NN
1338 \l_tmpa_seq
1339 \l_tmpa_tl
1340 \seq_pop:NN
1341 \l_tmpa_seq
1342 \l_tmpb_tl
1343 }
1344 {
1345 \tl_set:Nn
1346 \l_tmpa_tl
1347 { ##1 }
1348 \tl_set:Nn
1349 \l_tmpb_tl
1350 { ##1 }
1351 }
1352 \tl_put_left:Nn
1353 \l_tmpa_tl
1354 { / }
1355 \tl_put_left:NV
1356 \l_tmpa_tl
1357 \l_@@_import_current_theme_tl
1358 \@@_setup_snippet:Vx
1359 \l_tmpb_tl
1360 { snippet = { \l_tmpa_tl } }

```

```

1361 }
Here, we load the theme.
1362 \@@_set_theme:V
1363 \l_@@_import_current_theme_tl
1364 },
1365 }
1366 \cs_generate_variant:Nn
1367 \tl_replace_all:Nnn
1368 { NVn }
1369 \cs_generate_variant:Nn
1370 \@@_set_theme:n
1371 { V }
1372 \cs_generate_variant:Nn
1373 \@@_setup_snippet:nn
1374 { Vx }

```

## 2.2.5 Token Renderers

The following  $\TeX$  macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.6).

To enable the enumeration of token renderers, we will maintain the `\g_@@_renderers_seq` sequence.

```

1375 \ExplSyntaxOn
1376 \seq_new:N \g_@@_renderers_seq

```

To enable the reflection of token renderers and their parameters, we will maintain the `\g_@@_renderer_arities_prop` property list.

```

1377 \prop_new:N \g_@@_renderer_arities_prop
1378 \ExplSyntaxOff

```

### 2.2.5.1 Attribute Renderers

The following macros are only produced, when at least one of the following options for markdown attributes on different elements is enabled:

- `autoIdentifiers`
- `fencedCodeAttributes`
- `gfmAutoIdentifiers`
- `headerAttributes`
- `inlineCodeAttributes`
- `linkAttributes`

`\markdownRendererAttributeIdentifier` represents the  $\langle identifier \rangle$  of a markdown element (`id="⟨identifier⟩"` in HTML and `#⟨identifier⟩` in markdown attributes). The macro receives a single attribute that corresponds to the  $\langle identifier \rangle$ .

`\markdownRendererAttributeClassName` represents the  $\langle class name \rangle$  of a markdown element (`class="⟨class name⟩ ..."` in HTML and `.⟨class name⟩` in markdown attributes). The macro receives a single attribute that corresponds to the  $\langle class name \rangle$ .

`\markdownRendererAttributeKeyValue` represents a HTML attribute in the form  $\langle key \rangle = \langle value \rangle$  that is neither an identifier nor a class name. The macro receives two attributes that correspond to the  $\langle key \rangle$  and the  $\langle value \rangle$ , respectively.

```

1379 \def\markdownRendererAttributeIdentifier{%
1380 \markdownRendererAttributeIdentifierPrototype}%
1381 \ExplSyntaxOn
1382 \seq_gput_right:Nn
1383 \g_@@_renderers_seq
1384 { attributeIdentifier }
1385 \prop_gput:Nnn
1386 \g_@@_renderer_arities_prop
1387 { attributeIdentifier }
1388 { 1 }
1389 \ExplSyntaxOff
1390 \def\markdownRendererAttributeClassName{%
1391 \markdownRendererAttributeClassNamePrototype}%
1392 \ExplSyntaxOn
1393 \seq_gput_right:Nn
1394 \g_@@_renderers_seq
1395 { attributeClassName }
1396 \prop_gput:Nnn
1397 \g_@@_renderer_arities_prop
1398 { attributeClassName }
1399 { 1 }
1400 \ExplSyntaxOff
1401 \def\markdownRendererAttributeKeyValue{%
1402 \markdownRendererAttributeKeyValuePrototype}%
1403 \ExplSyntaxOn
1404 \seq_gput_right:Nn
1405 \g_@@_renderers_seq
1406 { attributeKeyValue }
1407 \prop_gput:Nnn
1408 \g_@@_renderer_arities_prop
1409 { attributeKeyValue }
1410 { 2 }
1411 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers



The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```
1412 \def\markdownRendererBlockQuoteBegin{%
1413 \markdownRendererBlockQuoteBeginPrototype}%
1414 \ExplSyntaxOn
1415 \seq_gput_right:Nn
1416 \g_@@_renderers_seq
1417 { blockQuoteBegin }
1418 \prop_gput:Nnn
1419 \g_@@_renderer_arities_prop
1420 { blockQuoteBegin }
1421 { 0 }
1422 \ExplSyntaxOff
```

The `\markdownRendererBlockQuoteEnd` macro represents the end of a block quote. The macro receives no arguments.

```
1423 \def\markdownRendererBlockQuoteEnd{%
1424 \markdownRendererBlockQuoteEndPrototype}%
1425 \ExplSyntaxOn
1426 \seq_gput_right:Nn
1427 \g_@@_renderers_seq
1428 { blockQuoteEnd }
1429 \prop_gput:Nnn
1430 \g_@@_renderer_arities_prop
1431 { blockQuoteEnd }
1432 { 0 }
1433 \ExplSyntaxOff
```

### 2.2.5.3 Bracketed Spans Attribute Context Renderers

The following macros are only produced, when the `bracketedSpans` option is enabled.

The `\markdownRendererBracketedSpanAttributeContextBegin` and `\markdownRendererBracketedSpanAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of an inline bracketed span apply. The macros receive no arguments.

```
1434 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1435 \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1436 \ExplSyntaxOn
1437 \seq_gput_right:Nn
1438 \g_@@_renderers_seq
1439 { bracketedSpanAttributeContextBegin }
1440 \prop_gput:Nnn
1441 \g_@@_renderer_arities_prop
1442 { bracketedSpanAttributeContextBegin }
1443 { 0 }
1444 \ExplSyntaxOff
```

```

1445 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1446 \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1447 \ExplSyntaxOn
1448 \seq_gput_right:Nn
1449 \g_@@_renderers_seq
1450 { bracketedSpanAttributeContextEnd }
1451 \prop_gput:Nnn
1452 \g_@@_renderer_arities_prop
1453 { bracketedSpanAttributeContextEnd }
1454 { 0 }
1455 \ExplSyntaxOff

```

#### 2.2.5.4 Bullet List Renderers

The `\markdownRendererUlBegin` macro represents the beginning of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

1456 \def\markdownRendererUlBegin{%
1457 \markdownRendererUlBeginPrototype}%
1458 \ExplSyntaxOn
1459 \seq_gput_right:Nn
1460 \g_@@_renderers_seq
1461 { ulBegin }
1462 \prop_gput:Nnn
1463 \g_@@_renderer_arities_prop
1464 { ulBegin }
1465 { 0 }
1466 \ExplSyntaxOff

```

The `\markdownRendererUlBeginTight` macro represents the beginning of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```

1467 \def\markdownRendererUlBeginTight{%
1468 \markdownRendererUlBeginTightPrototype}%
1469 \ExplSyntaxOn
1470 \seq_gput_right:Nn
1471 \g_@@_renderers_seq
1472 { ulBeginTight }
1473 \prop_gput:Nnn
1474 \g_@@_renderer_arities_prop
1475 { ulBeginTight }
1476 { 0 }
1477 \ExplSyntaxOff

```

The `\markdownRendererUlItem` macro represents an item in a bulleted list. The macro receives no arguments.

```

1478 \def\markdownRendererUListItem{%
1479 \markdownRendererUListItemPrototype}%
1480 \ExplSyntaxOn
1481 \seq_gput_right:Nn
1482 \g_@@_renderers_seq
1483 { ulItem }
1484 \prop_gput:Nnn
1485 \g_@@_renderer_arities_prop
1486 { ulItem }
1487 { 0 }
1488 \ExplSyntaxOff

```

The `\markdownRendererUListItemEnd` macro represents the end of an item in a bulleted list. The macro receives no arguments.

```

1489 \def\markdownRendererUListItemEnd{%
1490 \markdownRendererUListItemEndPrototype}%
1491 \ExplSyntaxOn
1492 \seq_gput_right:Nn
1493 \g_@@_renderers_seq
1494 { ulItemEnd }
1495 \prop_gput:Nnn
1496 \g_@@_renderer_arities_prop
1497 { ulItemEnd }
1498 { 0 }
1499 \ExplSyntaxOff

```

The `\markdownRendererUListEnd` macro represents the end of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

1500 \def\markdownRendererUListEnd{%
1501 \markdownRendererUListEndPrototype}%
1502 \ExplSyntaxOn
1503 \seq_gput_right:Nn
1504 \g_@@_renderers_seq
1505 { ulEnd }
1506 \prop_gput:Nnn
1507 \g_@@_renderer_arities_prop
1508 { ulEnd }
1509 { 0 }
1510 \ExplSyntaxOff

```

The `\markdownRendererUListEndTight` macro represents the end of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```

1511 \def\markdownRendererUListEndTight{%

```

```

1512 \markdownRendererUlnEndTightPrototype}%
1513 \ExplSyntaxOn
1514 \seq_gput_right:Nn
1515 \g_@@_renderers_seq
1516 { ulEndTight }
1517 \prop_gput:Nnn
1518 \g_@@_renderer_arities_prop
1519 { ulEndTight }
1520 { 0 }
1521 \ExplSyntaxOff

```

### 2.2.5.5 Citation Renderers

The `\markdownRendererCite` macro represents a string of one or more parenthetical citations. This macro will only be produced, when the  `citations`  option is enabled. The macro receives the parameter `{<number of citations>}` followed by `<suppress author> {<prenote>}{<postnote>}{<name>}` repeated `<number of citations>` times. The `<suppress author>` parameter is either the token `-`, when the author's name is to be suppressed, or `+` otherwise.

```

1522 \def\markdownRendererCite{%
1523 \markdownRendererCitePrototype}%
1524 \ExplSyntaxOn
1525 \seq_gput_right:Nn
1526 \g_@@_renderers_seq
1527 { cite }
1528 \prop_gput:Nnn
1529 \g_@@_renderer_arities_prop
1530 { cite }
1531 { 1 }
1532 \ExplSyntaxOff

```

The `\markdownRendererTextCite` macro represents a string of one or more text citations. This macro will only be produced, when the  `citations`  option is enabled. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

```

1533 \def\markdownRendererTextCite{%
1534 \markdownRendererTextCitePrototype}%
1535 \ExplSyntaxOn
1536 \seq_gput_right:Nn
1537 \g_@@_renderers_seq
1538 { textCite }
1539 \prop_gput:Nnn
1540 \g_@@_renderer_arities_prop
1541 { textCite }
1542 { 1 }
1543 \ExplSyntaxOff

```

### 2.2.5.6 Code Block Renderers

The `\markdownRendererInputVerbatim` macro represents a code block. The macro receives a single argument that corresponds to the filename of a file containing the code block contents.

```
1544 \def\markdownRendererInputVerbatim{%
1545 \markdownRendererInputVerbatimPrototype}%
1546 \ExplSyntaxOn
1547 \seq_gput_right:Nn
1548 \g_@@_renderers_seq
1549 { inputVerbatim }
1550 \prop_gput:Nnn
1551 \g_@@_renderer_arities_prop
1552 { inputVerbatim }
1553 { 1 }
1554 \ExplSyntaxOff
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives three arguments that correspond to the filename of a file containing the code block contents, the fully escaped code fence infostring that can be directly typeset, and the raw code fence infostring that can be used outside typesetting.

```
1555 \def\markdownRendererInputFencedCode{%
1556 \markdownRendererInputFencedCodePrototype}%
1557 \ExplSyntaxOn
1558 \seq_gput_right:Nn
1559 \g_@@_renderers_seq
1560 { inputFencedCode }
1561 \prop_gput:Nnn
1562 \g_@@_renderer_arities_prop
1563 { inputFencedCode }
1564 { 3 }
1565 \ExplSyntaxOff
```

### 2.2.5.7 Code Span Renderer

The `\markdownRendererCodeSpan` macro represents inline code span in the input text. It receives a single argument that corresponds to the inline code span.

```
1566 \def\markdownRendererCodeSpan{%
1567 \markdownRendererCodeSpanPrototype}%
1568 \ExplSyntaxOn
1569 \seq_gput_right:Nn
1570 \g_@@_renderers_seq
1571 { codeSpan }
1572 \prop_gput:Nnn
1573 \g_@@_renderer_arities_prop
1574 { codeSpan }
```

```
1575 { 1 }
1576 \ExplSyntaxOff
```

### 2.2.5.8 Code Span Attribute Context Renderers

The following macros are only produced, when the `inlineCodeAttributes` option is enabled.

The `\markdownRendererCodeSpanAttributeContextBegin` and `\markdownRendererCodeSpanAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of an inline code span apply. The macros receive no arguments.

```
1577 \def\markdownRendererCodeSpanAttributeContextBegin{%
1578 \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1579 \ExplSyntaxOn
1580 \seq_gput_right:Nn
1581 \g_@@_renderers_seq
1582 { codeSpanAttributeContextBegin }
1583 \prop_gput:Nnn
1584 \g_@@_renderer_arities_prop
1585 { codeSpanAttributeContextBegin }
1586 { 0 }
1587 \ExplSyntaxOff
1588 \def\markdownRendererCodeSpanAttributeContextEnd{%
1589 \markdownRendererCodeSpanAttributeContextEndPrototype}%
1590 \ExplSyntaxOn
1591 \seq_gput_right:Nn
1592 \g_@@_renderers_seq
1593 { codeSpanAttributeContextEnd }
1594 \prop_gput:Nnn
1595 \g_@@_renderer_arities_prop
1596 { codeSpanAttributeContextEnd }
1597 { 0 }
1598 \ExplSyntaxOff
```

### 2.2.5.9 Content Block Renderers

The `\markdownRendererContentBlock` macro represents an iA Writer content block. It receives four arguments: the local file or online image filename extension cast to the lower case, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

```
1599 \def\markdownRendererContentBlock{%
1600 \markdownRendererContentBlockPrototype}%
1601 \ExplSyntaxOn
1602 \seq_gput_right:Nn
1603 \g_@@_renderers_seq
1604 { contentBlock }
1605 \prop_gput:Nnn
```

```

1606 \g_@@_renderer_arities_prop
1607 { contentBlock }
1608 { 4 }
1609 \ExplSyntaxOff

```

The `\markdownRendererContentBlockOnlineImage` macro represents an iA Writer online image content block. The macro receives the same arguments as `\markdownRendererContentBlock`.

```

1610 \def\markdownRendererContentBlockOnlineImage{%
1611 \markdownRendererContentBlockOnlineImagePrototype}%
1612 \ExplSyntaxOn
1613 \seq_gput_right:Nn
1614 \g_@@_renderers_seq
1615 { contentBlockOnlineImage }
1616 \prop_gput:Nnn
1617 \g_@@_renderer_arities_prop
1618 { contentBlockOnlineImage }
1619 { 4 }
1620 \ExplSyntaxOff

```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension  $s$ . If any `markdown-languages.json` file found by `kpathsea`<sup>31</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s, s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local T<sub>E</sub>X directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```

1621 \def\markdownRendererContentBlockCode{%
1622 \markdownRendererContentBlockCodePrototype}%
1623 \ExplSyntaxOn
1624 \seq_gput_right:Nn
1625 \g_@@_renderers_seq
1626 { contentBlockCode }
1627 \prop_gput:Nnn
1628 \g_@@_renderer_arities_prop

```

---

<sup>31</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

1629 { contentBlockCode }
1630 { 5 }
1631 \ExplSyntaxOff

```

### 2.2.5.10 Definition List Renderers

The following macros are only produced, when the `definitionLists` option is enabled.

The `\markdownRendererDlBegin` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

1632 \def\markdownRendererDlBegin{%
1633 \markdownRendererDlBeginPrototype}%
1634 \ExplSyntaxOn
1635 \seq_gput_right:Nn
1636 \g_@@_renderers_seq
1637 { dlBegin }
1638 \prop_gput:Nnn
1639 \g_@@_renderer_arities_prop
1640 { dlBegin }
1641 { 0 }
1642 \ExplSyntaxOff

```

The `\markdownRendererDlBeginTight` macro represents the beginning of a definition list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```

1643 \def\markdownRendererDlBeginTight{%
1644 \markdownRendererDlBeginTightPrototype}%
1645 \ExplSyntaxOn
1646 \seq_gput_right:Nn
1647 \g_@@_renderers_seq
1648 { dlBeginTight }
1649 \prop_gput:Nnn
1650 \g_@@_renderer_arities_prop
1651 { dlBeginTight }
1652 { 0 }
1653 \ExplSyntaxOff

```

The `\markdownRendererDlItem` macro represents a term in a definition list. The macro receives a single argument that corresponds to the term being defined.

```

1654 \def\markdownRendererDlItem{%
1655 \markdownRendererDlItemPrototype}%
1656 \ExplSyntaxOn
1657 \seq_gput_right:Nn
1658 \g_@@_renderers_seq

```



```

1659 { dlItem }
1660 \prop_gput:Nnn
1661 \g_@@_renderer_arities_prop
1662 { dlItem }
1663 { 1 }
1664 \ExplSyntaxOff

```

The `\markdownRendererDlItemEnd` macro represents the end of a list of definitions for a single term.

```

1665 \def\markdownRendererDlItemEnd{%
1666 \markdownRendererDlItemEndPrototype}%
1667 \ExplSyntaxOn
1668 \seq_gput_right:Nn
1669 \g_@@_renderers_seq
1670 { dlItemEnd }
1671 \prop_gput:Nnn
1672 \g_@@_renderer_arities_prop
1673 { dlItemEnd }
1674 { 0 }
1675 \ExplSyntaxOff

```

The `\markdownRendererDlDefinitionBegin` macro represents the beginning of a definition in a definition list. There can be several definitions for a single term.

```

1676 \def\markdownRendererDlDefinitionBegin{%
1677 \markdownRendererDlDefinitionBeginPrototype}%
1678 \ExplSyntaxOn
1679 \seq_gput_right:Nn
1680 \g_@@_renderers_seq
1681 { dlDefinitionBegin }
1682 \prop_gput:Nnn
1683 \g_@@_renderer_arities_prop
1684 { dlDefinitionBegin }
1685 { 0 }
1686 \ExplSyntaxOff

```

The `\markdownRendererDlDefinitionEnd` macro represents the end of a definition in a definition list. There can be several definitions for a single term.

```

1687 \def\markdownRendererDlDefinitionEnd{%
1688 \markdownRendererDlDefinitionEndPrototype}%
1689 \ExplSyntaxOn
1690 \seq_gput_right:Nn
1691 \g_@@_renderers_seq
1692 { dlDefinitionEnd }
1693 \prop_gput:Nnn
1694 \g_@@_renderer_arities_prop
1695 { dlDefinitionEnd }
1696 { 0 }

```

```
1697 \ExplSyntaxOff
```

The `\markdownRendererDlEnd` macro represents the end of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
1698 \def\markdownRendererDlEnd{%
1699 \markdownRendererDlEndPrototype}%
1700 \ExplSyntaxOn
1701 \seq_gput_right:Nn
1702 \g_@@_renderers_seq
1703 { dlEnd }
1704 \prop_gput:Nnn
1705 \g_@@_renderer_arities_prop
1706 { dlEnd }
1707 { 0 }
1708 \ExplSyntaxOff
```

The `\markdownRendererDlEndTight` macro represents the end of a definition list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```
1709 \def\markdownRendererDlEndTight{%
1710 \markdownRendererDlEndTightPrototype}%
1711 \ExplSyntaxOn
1712 \seq_gput_right:Nn
1713 \g_@@_renderers_seq
1714 { dlEndTight }
1715 \prop_gput:Nnn
1716 \g_@@_renderer_arities_prop
1717 { dlEndTight }
1718 { 0 }
1719 \ExplSyntaxOff
```

### 2.2.5.11 Ellipsis Renderer

The `\markdownRendererEllipsis` macro replaces any occurrence of ASCII ellipses in the input text. This macro will only be produced, when the `smartEllipses` option is enabled. The macro receives no arguments.

```
1720 \def\markdownRendererEllipsis{%
1721 \markdownRendererEllipsisPrototype}%
1722 \ExplSyntaxOn
1723 \seq_gput_right:Nn
1724 \g_@@_renderers_seq
1725 { ellipsis }
1726 \prop_gput:Nnn
1727 \g_@@_renderer_arities_prop
```

```

1728 { ellipsis }
1729 { 0 }
1730 \ExplSyntaxOff

```

### 2.2.5.12 Emphasis Renderers

The `\markdownRendererEmphasis` macro represents an emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

1731 \def\markdownRendererEmphasis{%
1732 \markdownRendererEmphasisPrototype}%
1733 \ExplSyntaxOn
1734 \seq_gput_right:Nn
1735 \g_@@_renderers_seq
1736 { emphasis }
1737 \prop_gput:Nnn
1738 \g_@@_renderer_arities_prop
1739 { emphasis }
1740 { 1 }
1741 \ExplSyntaxOff

```

The `\markdownRendererStrongEmphasis` macro represents a strongly emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

1742 \def\markdownRendererStrongEmphasis{%
1743 \markdownRendererStrongEmphasisPrototype}%
1744 \ExplSyntaxOn
1745 \seq_gput_right:Nn
1746 \g_@@_renderers_seq
1747 { strongEmphasis }
1748 \prop_gput:Nnn
1749 \g_@@_renderer_arities_prop
1750 { strongEmphasis }
1751 { 1 }
1752 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

The following macros are only produced, when the `fencedCode` option is enabled.

The `\markdownRendererFencedCodeAttributeContextBegin` and `\markdownRendererFencedCodeAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a fenced code apply. The macros receive no arguments.

```

1753 \def\markdownRendererFencedCodeAttributeContextBegin{%
1754 \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1755 \ExplSyntaxOn
1756 \seq_gput_right:Nn

```

```

1757 \g_@@_renderers_seq
1758 { fencedCodeAttributeContextBegin }
1759 \prop_gput:Nnn
1760 \g_@@_renderer_arities_prop
1761 { fencedCodeAttributeContextBegin }
1762 { 0 }
1763 \ExplSyntaxOff
1764 \def\markdownRendererFencedCodeAttributeContextEnd{%
1765 \markdownRendererFencedCodeAttributeContextEndPrototype}%
1766 \ExplSyntaxOn
1767 \seq_gput_right:Nn
1768 \g_@@_renderers_seq
1769 { fencedCodeAttributeContextEnd }
1770 \prop_gput:Nnn
1771 \g_@@_renderer_arities_prop
1772 { fencedCodeAttributeContextEnd }
1773 { 0 }
1774 \ExplSyntaxOff

```

#### 2.2.5.14 Fenced Div Attribute Context Renderers

The following macros are only produced, when the `fencedDiv` option is enabled.

The `\markdownRendererFencedDivAttributeContextBegin` and `\markdownRendererFencedDivAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a div apply. The macros receive no arguments.

```

1775 \def\markdownRendererFencedDivAttributeContextBegin{%
1776 \markdownRendererFencedDivAttributeContextBeginPrototype}%
1777 \ExplSyntaxOn
1778 \seq_gput_right:Nn
1779 \g_@@_renderers_seq
1780 { fencedDivAttributeContextBegin }
1781 \prop_gput:Nnn
1782 \g_@@_renderer_arities_prop
1783 { fencedDivAttributeContextBegin }
1784 { 0 }
1785 \ExplSyntaxOff
1786 \def\markdownRendererFencedDivAttributeContextEnd{%
1787 \markdownRendererFencedDivAttributeContextEndPrototype}%
1788 \ExplSyntaxOn
1789 \seq_gput_right:Nn
1790 \g_@@_renderers_seq
1791 { fencedDivAttributeContextEnd }
1792 \prop_gput:Nnn
1793 \g_@@_renderer_arities_prop
1794 { fencedDivAttributeContextEnd }
1795 { 0 }
1796 \ExplSyntaxOff

```

### 2.2.5.15 Header Attribute Context Renderers

The following macros are only produced, when the `autoIdentifiers`, `gfmAutoIdentifiers`, or `headerAttributes` options are enabled.

The `\markdownRendererHeaderAttributeContextBegin` and `\markdownRendererHeaderAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a heading apply. The macros receive no arguments.

```
1797 \def\markdownRendererHeaderAttributeContextBegin{%
1798 \markdownRendererHeaderAttributeContextBeginPrototype}%
1799 \ExplSyntaxOn
1800 \seq_gput_right:Nn
1801 \g_@@_renderers_seq
1802 { headerAttributeContextBegin }
1803 \prop_gput:Nnn
1804 \g_@@_renderer_arities_prop
1805 { headerAttributeContextBegin }
1806 { 0 }
1807 \ExplSyntaxOff
1808 \def\markdownRendererHeaderAttributeContextEnd{%
1809 \markdownRendererHeaderAttributeContextEndPrototype}%
1810 \ExplSyntaxOn
1811 \seq_gput_right:Nn
1812 \g_@@_renderers_seq
1813 { headerAttributeContextEnd }
1814 \prop_gput:Nnn
1815 \g_@@_renderer_arities_prop
1816 { headerAttributeContextEnd }
1817 { 0 }
1818 \ExplSyntaxOff
```

### 2.2.5.16 Heading Renderers

The `\markdownRendererHeadingOne` macro represents a first level heading. The macro receives a single argument that corresponds to the heading text.

```
1819 \def\markdownRendererHeadingOne{%
1820 \markdownRendererHeadingOnePrototype}%
1821 \ExplSyntaxOn
1822 \seq_gput_right:Nn
1823 \g_@@_renderers_seq
1824 { headingOne }
1825 \prop_gput:Nnn
1826 \g_@@_renderer_arities_prop
1827 { headingOne }
1828 { 1 }
1829 \ExplSyntaxOff
```

The `\markdownRendererHeadingTwo` macro represents a second level heading. The macro receives a single argument that corresponds to the heading text.

```
1830 \def\markdownRendererHeadingTwo{%
1831 \markdownRendererHeadingTwoPrototype}%
1832 \ExplSyntaxOn
1833 \seq_gput_right:Nn
1834 \g_@@_renderers_seq
1835 { headingTwo }
1836 \prop_gput:Nnn
1837 \g_@@_renderer_arities_prop
1838 { headingTwo }
1839 { 1 }
1840 \ExplSyntaxOff
```

The `\markdownRendererHeadingThree` macro represents a third level heading. The macro receives a single argument that corresponds to the heading text.

```
1841 \def\markdownRendererHeadingThree{%
1842 \markdownRendererHeadingThreePrototype}%
1843 \ExplSyntaxOn
1844 \seq_gput_right:Nn
1845 \g_@@_renderers_seq
1846 { headingThree }
1847 \prop_gput:Nnn
1848 \g_@@_renderer_arities_prop
1849 { headingThree }
1850 { 1 }
1851 \ExplSyntaxOff
```

The `\markdownRendererHeadingFour` macro represents a fourth level heading. The macro receives a single argument that corresponds to the heading text.

```
1852 \def\markdownRendererHeadingFour{%
1853 \markdownRendererHeadingFourPrototype}%
1854 \ExplSyntaxOn
1855 \seq_gput_right:Nn
1856 \g_@@_renderers_seq
1857 { headingFour }
1858 \prop_gput:Nnn
1859 \g_@@_renderer_arities_prop
1860 { headingFour }
1861 { 1 }
1862 \ExplSyntaxOff
```

The `\markdownRendererHeadingFive` macro represents a fifth level heading. The macro receives a single argument that corresponds to the heading text.

```
1863 \def\markdownRendererHeadingFive{%
1864 \markdownRendererHeadingFivePrototype}%
```

```

1865 \ExplSyntaxOn
1866 \seq_gput_right:Nn
1867 \g_@@_renderers_seq
1868 { headingFive }
1869 \prop_gput:Nnn
1870 \g_@@_renderer_arities_prop
1871 { headingFive }
1872 { 1 }
1873 \ExplSyntaxOff

```

The `\markdownRendererHeadingSix` macro represents a sixth level heading. The macro receives a single argument that corresponds to the heading text.

```

1874 \def\markdownRendererHeadingSix{%
1875 \markdownRendererHeadingSixPrototype}%
1876 \ExplSyntaxOn
1877 \seq_gput_right:Nn
1878 \g_@@_renderers_seq
1879 { headingSix }
1880 \prop_gput:Nnn
1881 \g_@@_renderer_arities_prop
1882 { headingSix }
1883 { 1 }
1884 \ExplSyntaxOff

```

#### 2.2.5.17 Inline HTML Comment Renderer

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

```

1885 \def\markdownRendererInlineHtmlComment{%
1886 \markdownRendererInlineHtmlCommentPrototype}%
1887 \ExplSyntaxOn
1888 \seq_gput_right:Nn
1889 \g_@@_renderers_seq
1890 { inlineHtmlComment }
1891 \prop_gput:Nnn
1892 \g_@@_renderer_arities_prop
1893 { inlineHtmlComment }
1894 { 1 }
1895 \ExplSyntaxOff

```

#### 2.2.5.18 HTML Tag and Element Renderers

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is

enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

The `\markdownRendererInputBlockHtmlElement` macro represents a block HTML element. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that filename of a file containing the contents of the HTML element.

```
1896 \def\markdownRendererInlineHtmlTag{%
1897 \markdownRendererInlineHtmlTagPrototype}%
1898 \ExplSyntaxOn
1899 \seq_gput_right:Nn
1900 \g_@@_renderers_seq
1901 { inlineHtmlTag }
1902 \prop_gput:Nnn
1903 \g_@@_renderer_arities_prop
1904 { inlineHtmlTag }
1905 { 1 }
1906 \ExplSyntaxOff
1907 \def\markdownRendererInputBlockHtmlElement{%
1908 \markdownRendererInputBlockHtmlElementPrototype}%
1909 \ExplSyntaxOn
1910 \seq_gput_right:Nn
1911 \g_@@_renderers_seq
1912 { inputBlockHtmlElement }
1913 \prop_gput:Nnn
1914 \g_@@_renderer_arities_prop
1915 { inputBlockHtmlElement }
1916 { 1 }
1917 \ExplSyntaxOff
```

### 2.2.5.19 Image Renderer

The `\markdownRendererImage` macro represents an image. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```
1918 \def\markdownRendererImage{%
1919 \markdownRendererImagePrototype}%
1920 \ExplSyntaxOn
1921 \seq_gput_right:Nn
1922 \g_@@_renderers_seq
1923 { image }
1924 \prop_gput:Nnn
1925 \g_@@_renderer_arities_prop
1926 { image }
1927 { 4 }
1928 \ExplSyntaxOff
```



### 2.2.5.20 Image Attribute Context Renderers

The following macros are only produced, when the `linkAttributes` option is enabled.

The `\markdownRendererImageAttributeContextBegin` and `\markdownRendererImageAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of an image apply. The macros receive no arguments.

```
1929 \def\markdownRendererImageAttributeContextBegin{%
1930 \markdownRendererImageAttributeContextBeginPrototype}%
1931 \ExplSyntaxOn
1932 \seq_gput_right:Nn
1933 \g_@@_renderers_seq
1934 { imageAttributeContextBegin }
1935 \prop_gput:Nnn
1936 \g_@@_renderer_arities_prop
1937 { imageAttributeContextBegin }
1938 { 0 }
1939 \ExplSyntaxOff
1940 \def\markdownRendererImageAttributeContextEnd{%
1941 \markdownRendererImageAttributeContextEndPrototype}%
1942 \ExplSyntaxOn
1943 \seq_gput_right:Nn
1944 \g_@@_renderers_seq
1945 { imageAttributeContextEnd }
1946 \prop_gput:Nnn
1947 \g_@@_renderer_arities_prop
1948 { imageAttributeContextEnd }
1949 { 0 }
1950 \ExplSyntaxOff
```

### 2.2.5.21 Interblock Separator Renderers

The `\markdownRendererInterblockSeparator` macro represents an interblock separator between two markdown block elements. The macro receives no arguments.

```
1951 \def\markdownRendererInterblockSeparator{%
1952 \markdownRendererInterblockSeparatorPrototype}%
1953 \ExplSyntaxOn
1954 \seq_gput_right:Nn
1955 \g_@@_renderers_seq
1956 { interblockSeparator }
1957 \prop_gput:Nnn
1958 \g_@@_renderer_arities_prop
1959 { interblockSeparator }
1960 { 0 }
1961 \ExplSyntaxOff
```

Users can use more than one blank line to delimit two block to indicate the end of a series of blocks that make up a logical paragraph. This produces a paragraph separator instead of an interblock separator. Between some blocks, such as markdown paragraphs, a paragraph separator is always produced.

The `\markdownRendererParagraphSeparator` macro represents a paragraph separator. The macro receives no arguments.

```

1962 \def\markdownRendererParagraphSeparator{%
1963 \markdownRendererParagraphSeparatorPrototype}%
1964 \ExplSyntaxOn
1965 \seq_gput_right:Nn
1966 \g_@@_renderers_seq
1967 { paragraphSeparator }
1968 \prop_gput:Nnn
1969 \g_@@_renderer_arities_prop
1970 { paragraphSeparator }
1971 { 0 }
1972 \ExplSyntaxOff

```

### 2.2.5.22 Line Block Renderers

The following macros are only produced, when the `lineBlocks` option is enabled.

The `\markdownRendererLineBlockBegin` and `\markdownRendererLineBlockEnd` macros represent the beginning and the end of a line block. The macros receive no arguments.

```

1973 \def\markdownRendererLineBlockBegin{%
1974 \markdownRendererLineBlockBeginPrototype}%
1975 \ExplSyntaxOn
1976 \seq_gput_right:Nn
1977 \g_@@_renderers_seq
1978 { lineBlockBegin }
1979 \prop_gput:Nnn
1980 \g_@@_renderer_arities_prop
1981 { lineBlockBegin }
1982 { 0 }
1983 \ExplSyntaxOff
1984 \def\markdownRendererLineBlockEnd{%
1985 \markdownRendererLineBlockEndPrototype}%
1986 \ExplSyntaxOn
1987 \seq_gput_right:Nn
1988 \g_@@_renderers_seq
1989 { lineBlockEnd }
1990 \prop_gput:Nnn
1991 \g_@@_renderer_arities_prop
1992 { lineBlockEnd }
1993 { 0 }
1994 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

The `\markdownRendererSoftLineBreak` macro represents a soft line break. The macro receives no arguments.

```
1995 \def\markdownRendererSoftLineBreak{%
1996 \markdownRendererSoftLineBreakPrototype}%
1997 \ExplSyntaxOn
1998 \seq_gput_right:Nn
1999 \g_@@_renderers_seq
2000 { softLineBreak }
2001 \prop_gput:Nnn
2002 \g_@@_renderer_arities_prop
2003 { softLineBreak }
2004 { 0 }
2005 \ExplSyntaxOff
```

The `\markdownRendererHardLineBreak` macro represents a hard line break. The macro receives no arguments.

```
2006 \def\markdownRendererHardLineBreak{%
2007 \markdownRendererHardLineBreakPrototype}%
2008 \ExplSyntaxOn
2009 \seq_gput_right:Nn
2010 \g_@@_renderers_seq
2011 { hardLineBreak }
2012 \prop_gput:Nnn
2013 \g_@@_renderer_arities_prop
2014 { hardLineBreak }
2015 { 0 }
2016 \ExplSyntaxOff
```

### 2.2.5.24 Link Renderer

The `\markdownRendererLink` macro represents a hyperlink. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```
2017 \def\markdownRendererLink{%
2018 \markdownRendererLinkPrototype}%
2019 \ExplSyntaxOn
2020 \seq_gput_right:Nn
2021 \g_@@_renderers_seq
2022 { link }
2023 \prop_gput:Nnn
2024 \g_@@_renderer_arities_prop
2025 { link }
2026 { 4 }
2027 \ExplSyntaxOff
```

### 2.2.5.25 Link Attribute Context Renderers

The following macros are only produced, when the `linkAttributes` option is enabled.

The `\markdownRendererLinkAttributeContextBegin` and `\markdownRendererLinkAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a hyperlink apply. The macros receive no arguments.

```
2028 \def\markdownRendererLinkAttributeContextBegin{%
2029 \markdownRendererLinkAttributeContextBeginPrototype}%
2030 \ExplSyntaxOn
2031 \seq_gput_right:Nn
2032 \g_@@_renderers_seq
2033 { linkAttributeContextBegin }
2034 \prop_gput:Nnn
2035 \g_@@_renderer_arities_prop
2036 { linkAttributeContextBegin }
2037 { 0 }
2038 \ExplSyntaxOff
2039 \def\markdownRendererLinkAttributeContextEnd{%
2040 \markdownRendererLinkAttributeContextEndPrototype}%
2041 \ExplSyntaxOn
2042 \seq_gput_right:Nn
2043 \g_@@_renderers_seq
2044 { linkAttributeContextEnd }
2045 \prop_gput:Nnn
2046 \g_@@_renderer_arities_prop
2047 { linkAttributeContextEnd }
2048 { 0 }
2049 \ExplSyntaxOff
```

### 2.2.5.26 Marked Text Renderer

The following macro is only produced, when the `mark` option is enabled.

The `\markdownRendererMark` macro represents a span of marked or highlighted text. The macro receives a single argument that corresponds to the marked text.

```
2050 \def\markdownRendererMark{%
2051 \markdownRendererMarkPrototype}%
2052 \ExplSyntaxOn
2053 \seq_gput_right:Nn
2054 \g_@@_renderers_seq
2055 { mark }
2056 \prop_gput:Nnn
2057 \g_@@_renderer_arities_prop
2058 { mark }
2059 { 1 }
2060 \ExplSyntaxOff
```

### 2.2.5.27 Markdown Document Renderers

The `\markdownRendererDocumentBegin` and `\markdownRendererDocumentEnd` macros represent the beginning and the end of a *markdown* document. The macros receive no arguments.

A  $\TeX$  document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```
2061 \def\markdownRendererDocumentBegin{%
2062 \markdownRendererDocumentBeginPrototype}%
2063 \ExplSyntaxOn
2064 \seq_gput_right:Nn
2065 \g_@@_renderers_seq
2066 { documentBegin }
2067 \prop_gput:Nnn
2068 \g_@@_renderer_arities_prop
2069 { documentBegin }
2070 { 0 }
2071 \ExplSyntaxOff
2072 \def\markdownRendererDocumentEnd{%
2073 \markdownRendererDocumentEndPrototype}%
2074 \ExplSyntaxOn
2075 \seq_gput_right:Nn
2076 \g_@@_renderers_seq
2077 { documentEnd }
2078 \prop_gput:Nnn
2079 \g_@@_renderer_arities_prop
2080 { documentEnd }
2081 { 0 }
2082 \ExplSyntaxOff
```

### 2.2.5.28 Non-Breaking Space Renderer

The `\markdownRendererNbsp` macro represents a non-breaking space.

```
2083 \def\markdownRendererNbsp{%
2084 \markdownRendererNbspPrototype}%
2085 \ExplSyntaxOn
2086 \seq_gput_right:Nn
2087 \g_@@_renderers_seq
2088 { nbsp }
2089 \prop_gput:Nnn
2090 \g_@@_renderer_arities_prop
2091 { nbsp }
2092 { 0 }
2093 \ExplSyntaxOff
```

### 2.2.5.29 Note Renderer

The `\markdownRendererNote` macro represents a note. This macro will only be produced, when the `notes` option is enabled. The macro receives a single argument that corresponds to the note text.

```
2094 \def\markdownRendererNote{%
2095 \markdownRendererNotePrototype}%
2096 \ExplSyntaxOn
2097 \seq_gput_right:Nn
2098 \g_@@_renderers_seq
2099 { note }
2100 \prop_gput:Nnn
2101 \g_@@_renderer_arities_prop
2102 { note }
2103 { 1 }
2104 \ExplSyntaxOff
```

### 2.2.5.30 Ordered List Renderers

The `\markdownRendererOlBegin` macro represents the beginning of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```
2105 \def\markdownRendererOlBegin{%
2106 \markdownRendererOlBeginPrototype}%
2107 \ExplSyntaxOn
2108 \seq_gput_right:Nn
2109 \g_@@_renderers_seq
2110 { olBegin }
2111 \prop_gput:Nnn
2112 \g_@@_renderer_arities_prop
2113 { olBegin }
2114 { 0 }
2115 \ExplSyntaxOff
```

The `\markdownRendererOlBeginTight` macro represents the beginning of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```
2116 \def\markdownRendererOlBeginTight{%
2117 \markdownRendererOlBeginTightPrototype}%
2118 \ExplSyntaxOn
2119 \seq_gput_right:Nn
2120 \g_@@_renderers_seq
2121 { olBeginTight }
2122 \prop_gput:Nnn
```

```

2123 \g_@@_renderer_arities_prop
2124 { olBeginTight }
2125 { 0 }
2126 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBegin` macro represents the beginning of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives two arguments: the style of the list item labels (`Decimal`, `LowerRoman`, `UpperRoman`, `LowerAlpha`, and `UpperAlpha`), and the style of delimiters between list item labels and texts (`Default`, `OneParen`, and `Period`).

```

2127 \def\markdownRendererFancyOlBegin{%
2128 \markdownRendererFancyOlBeginPrototype}%
2129 \ExplSyntaxOn
2130 \seq_gput_right:Nn
2131 \g_@@_renderers_seq
2132 { fancyOlBegin }
2133 \prop_gput:Nnn
2134 \g_@@_renderer_arities_prop
2135 { fancyOlBegin }
2136 { 2 }
2137 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBeginTight` macro represents the beginning of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives two arguments: the style of the list item labels, and the style of delimiters between list item labels and texts. See the `\markdownRendererFancyOlBegin` macro for the valid style values.

```

2138 \def\markdownRendererFancyOlBeginTight{%
2139 \markdownRendererFancyOlBeginTightPrototype}%
2140 \ExplSyntaxOn
2141 \seq_gput_right:Nn
2142 \g_@@_renderers_seq
2143 { fancyOlBeginTight }
2144 \prop_gput:Nnn
2145 \g_@@_renderer_arities_prop
2146 { fancyOlBeginTight }
2147 { 2 }
2148 \ExplSyntaxOff

```

The `\markdownRendererOlItem` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

2149 \def\markdownRendererOlItem{%

```

```

2150 \markdownRendererOlItemPrototype}%
2151 \ExplSyntaxOn
2152 \seq_gput_right:Nn
2153 \g_@@_renderers_seq
2154 { olItem }
2155 \prop_gput:Nnn
2156 \g_@@_renderer_arities_prop
2157 { olItem }
2158 { 0 }
2159 \ExplSyntaxOff

```

The `\markdownRendererOlItemEnd` macro represents the end of an item in an ordered list. This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

2160 \def\markdownRendererOlItemEnd{%
2161 \markdownRendererOlItemEndPrototype}%
2162 \ExplSyntaxOn
2163 \seq_gput_right:Nn
2164 \g_@@_renderers_seq
2165 { olItemEnd }
2166 \prop_gput:Nnn
2167 \g_@@_renderer_arities_prop
2168 { olItemEnd }
2169 { 0 }
2170 \ExplSyntaxOff

```

The `\markdownRendererOlItemWithNumber` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is enabled and the `fancyLists` option is disabled. The macro receives a single numeric argument that corresponds to the item number.

```

2171 \def\markdownRendererOlItemWithNumber{%
2172 \markdownRendererOlItemWithNumberPrototype}%
2173 \ExplSyntaxOn
2174 \seq_gput_right:Nn
2175 \g_@@_renderers_seq
2176 { olItemWithNumber }
2177 \prop_gput:Nnn
2178 \g_@@_renderer_arities_prop
2179 { olItemWithNumber }
2180 { 1 }
2181 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItem` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is enabled. The macro receives no arguments.

```

2182 \def\markdownRendererFancyOlItem{%

```



```

2183 \markdownRendererFancyO1ItemPrototype}%
2184 \ExplSyntaxOn
2185 \seq_gput_right:Nn
2186 \g_@@_renderers_seq
2187 { fancyO1Item }
2188 \prop_gput:Nnn
2189 \g_@@_renderer_arities_prop
2190 { fancyO1Item }
2191 { 0 }
2192 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemEnd` macro represents the end of an item in a fancy ordered list. This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

2193 \def\markdownRendererFancyO1ItemEnd{%
2194 \markdownRendererFancyO1ItemEndPrototype}%
2195 \ExplSyntaxOn
2196 \seq_gput_right:Nn
2197 \g_@@_renderers_seq
2198 { fancyO1ItemEnd }
2199 \prop_gput:Nnn
2200 \g_@@_renderer_arities_prop
2201 { fancyO1ItemEnd }
2202 { 0 }
2203 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemWithNumber` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` and `fancyLists` options are enabled. The macro receives a single numeric argument that corresponds to the item number.

```

2204 \def\markdownRendererFancyO1ItemWithNumber{%
2205 \markdownRendererFancyO1ItemWithNumberPrototype}%
2206 \ExplSyntaxOn
2207 \seq_gput_right:Nn
2208 \g_@@_renderers_seq
2209 { fancyO1ItemWithNumber }
2210 \prop_gput:Nnn
2211 \g_@@_renderer_arities_prop
2212 { fancyO1ItemWithNumber }
2213 { 1 }
2214 \ExplSyntaxOff

```

The `\markdownRendererO1End` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

2215 \def\markdownRendererO1End{%
2216 \markdownRendererO1EndPrototype}%
2217 \ExplSyntaxOn
2218 \seq_gput_right:Nn
2219 \g_@@_renderers_seq
2220 { olEnd }
2221 \prop_gput:Nnn
2222 \g_@@_renderer_arities_prop
2223 { olEnd }
2224 { 0 }
2225 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

2226 \def\markdownRendererO1EndTight{%
2227 \markdownRendererO1EndTightPrototype}%
2228 \ExplSyntaxOn
2229 \seq_gput_right:Nn
2230 \g_@@_renderers_seq
2231 { olEndTight }
2232 \prop_gput:Nnn
2233 \g_@@_renderer_arities_prop
2234 { olEndTight }
2235 { 0 }
2236 \ExplSyntaxOff

```

The `\markdownRendererFancyO1End` macro represents the end of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

2237 \def\markdownRendererFancyO1End{%
2238 \markdownRendererFancyO1EndPrototype}%
2239 \ExplSyntaxOn
2240 \seq_gput_right:Nn
2241 \g_@@_renderers_seq
2242 { fancyO1End }
2243 \prop_gput:Nnn
2244 \g_@@_renderer_arities_prop
2245 { fancyO1End }
2246 { 0 }
2247 \ExplSyntaxOff

```

The `\markdownRendererFancyO1EndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight).

This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```

2248 \def\markdownRendererFancy01EndTight{%
2249 \markdownRendererFancy01EndTightPrototype}%
2250 \ExplSyntaxOn
2251 \seq_gput_right:Nn
2252 \g_@@_renderers_seq
2253 { fancy01EndTight }
2254 \prop_gput:Nnn
2255 \g_@@_renderer_arities_prop
2256 { fancy01EndTight }
2257 { 0 }
2258 \ExplSyntaxOff

```

### 2.2.5.31 Raw Content Renderers

The `\markdownRendererInputRawInline` macro represents an inline raw span. The macro receives two arguments: the filename of a file containing the inline raw span contents and the raw attribute that designates the format of the inline raw span. This macro will only be produced, when the `rawAttribute` option is enabled.

```

2259 \def\markdownRendererInputRawInline{%
2260 \markdownRendererInputRawInlinePrototype}%
2261 \ExplSyntaxOn
2262 \seq_gput_right:Nn
2263 \g_@@_renderers_seq
2264 { inputRawInline }
2265 \prop_gput:Nnn
2266 \g_@@_renderer_arities_prop
2267 { inputRawInline }
2268 { 2 }
2269 \ExplSyntaxOff

```

The `\markdownRendererInputRawBlock` macro represents a raw block. The macro receives two arguments: the filename of a file containing the raw block and the raw attribute that designates the format of the raw block. This macro will only be produced, when the `rawAttribute` and `fencedCode` options are enabled.

```

2270 \def\markdownRendererInputRawBlock{%
2271 \markdownRendererInputRawBlockPrototype}%
2272 \ExplSyntaxOn
2273 \seq_gput_right:Nn
2274 \g_@@_renderers_seq
2275 { inputRawBlock }
2276 \prop_gput:Nnn
2277 \g_@@_renderer_arities_prop
2278 { inputRawBlock }
2279 { 2 }

```

```
2280 \ExplSyntaxOff
```

### 2.2.5.32 Section Renderers

The `\markdownRendererSectionBegin` and `\markdownRendererSectionEnd` macros represent the beginning and the end of a section based on headings.

```
2281 \def\markdownRendererSectionBegin{%
2282 \markdownRendererSectionBeginPrototype}%
2283 \ExplSyntaxOn
2284 \seq_gput_right:Nn
2285 \g_@@_renderers_seq
2286 { sectionBegin }
2287 \prop_gput:Nnn
2288 \g_@@_renderer_arities_prop
2289 { sectionBegin }
2290 { 0 }
2291 \ExplSyntaxOff
2292 \def\markdownRendererSectionEnd{%
2293 \markdownRendererSectionEndPrototype}%
2294 \ExplSyntaxOn
2295 \seq_gput_right:Nn
2296 \g_@@_renderers_seq
2297 { sectionEnd }
2298 \prop_gput:Nnn
2299 \g_@@_renderer_arities_prop
2300 { sectionEnd }
2301 { 0 }
2302 \ExplSyntaxOff
```

### 2.2.5.33 Replacement Character Renderers

The `\markdownRendererReplacementCharacter` macro represents the U+0000 and U+FFFD Unicode characters. The macro receives no arguments.

```
2303 \def\markdownRendererReplacementCharacter{%
2304 \markdownRendererReplacementCharacterPrototype}%
2305 \ExplSyntaxOn
2306 \seq_gput_right:Nn
2307 \g_@@_renderers_seq
2308 { replacementCharacter }
2309 \prop_gput:Nnn
2310 \g_@@_renderer_arities_prop
2311 { replacementCharacter }
2312 { 0 }
2313 \ExplSyntaxOff
```

### 2.2.5.34 Special Character Renderers

The following macros replace any special plain T<sub>E</sub>X characters, including the active pipe character (|) of ConT<sub>E</sub>Xt, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```

2314 \def\markdownRendererLeftBrace{%
2315 \markdownRendererLeftBracePrototype}%
2316 \ExplSyntaxOn
2317 \seq_gput_right:Nn
2318 \g_@@_renderers_seq
2319 { leftBrace }
2320 \prop_gput:Nnn
2321 \g_@@_renderer_arities_prop
2322 { leftBrace }
2323 { 0 }
2324 \ExplSyntaxOff
2325 \def\markdownRendererRightBrace{%
2326 \markdownRendererRightBracePrototype}%
2327 \ExplSyntaxOn
2328 \seq_gput_right:Nn
2329 \g_@@_renderers_seq
2330 { rightBrace }
2331 \prop_gput:Nnn
2332 \g_@@_renderer_arities_prop
2333 { rightBrace }
2334 { 0 }
2335 \ExplSyntaxOff
2336 \def\markdownRendererDollarSign{%
2337 \markdownRendererDollarSignPrototype}%
2338 \ExplSyntaxOn
2339 \seq_gput_right:Nn
2340 \g_@@_renderers_seq
2341 { dollarSign }
2342 \prop_gput:Nnn
2343 \g_@@_renderer_arities_prop
2344 { dollarSign }
2345 { 0 }
2346 \ExplSyntaxOff
2347 \def\markdownRendererPercentSign{%
2348 \markdownRendererPercentSignPrototype}%
2349 \ExplSyntaxOn
2350 \seq_gput_right:Nn
2351 \g_@@_renderers_seq
2352 { percentSign }
2353 \prop_gput:Nnn
2354 \g_@@_renderer_arities_prop
2355 { percentSign }
2356 { 0 }

```

```

2357 \ExplSyntaxOff
2358 \def\markdownRendererAmpersand{%
2359 \markdownRendererAmpersandPrototype}%
2360 \ExplSyntaxOn
2361 \seq_gput_right:Nn
2362 \g_@@_renderers_seq
2363 { ampersand }
2364 \prop_gput:Nnn
2365 \g_@@_renderer_arities_prop
2366 { ampersand }
2367 { 0 }
2368 \ExplSyntaxOff
2369 \def\markdownRendererUnderscore{%
2370 \markdownRendererUnderscorePrototype}%
2371 \ExplSyntaxOn
2372 \seq_gput_right:Nn
2373 \g_@@_renderers_seq
2374 { underscore }
2375 \prop_gput:Nnn
2376 \g_@@_renderer_arities_prop
2377 { underscore }
2378 { 0 }
2379 \ExplSyntaxOff
2380 \def\markdownRendererHash{%
2381 \markdownRendererHashPrototype}%
2382 \ExplSyntaxOn
2383 \seq_gput_right:Nn
2384 \g_@@_renderers_seq
2385 { hash }
2386 \prop_gput:Nnn
2387 \g_@@_renderer_arities_prop
2388 { hash }
2389 { 0 }
2390 \ExplSyntaxOff
2391 \def\markdownRendererCircumflex{%
2392 \markdownRendererCircumflexPrototype}%
2393 \ExplSyntaxOn
2394 \seq_gput_right:Nn
2395 \g_@@_renderers_seq
2396 { circumflex }
2397 \prop_gput:Nnn
2398 \g_@@_renderer_arities_prop
2399 { circumflex }
2400 { 0 }
2401 \ExplSyntaxOff
2402 \def\markdownRendererBackslash{%
2403 \markdownRendererBackslashPrototype}%

```

```

2404 \ExplSyntaxOn
2405 \seq_gput_right:Nn
2406 \g_@@_renderers_seq
2407 { backslash }
2408 \prop_gput:Nnn
2409 \g_@@_renderer_arities_prop
2410 { backslash }
2411 { 0 }
2412 \ExplSyntaxOff
2413 \def\markdownRendererTilde{%
2414 \markdownRendererTildePrototype}%
2415 \ExplSyntaxOn
2416 \seq_gput_right:Nn
2417 \g_@@_renderers_seq
2418 { tilde }
2419 \prop_gput:Nnn
2420 \g_@@_renderer_arities_prop
2421 { tilde }
2422 { 0 }
2423 \ExplSyntaxOff
2424 \def\markdownRendererPipe{%
2425 \markdownRendererPipePrototype}%
2426 \ExplSyntaxOn
2427 \seq_gput_right:Nn
2428 \g_@@_renderers_seq
2429 { pipe }
2430 \prop_gput:Nnn
2431 \g_@@_renderer_arities_prop
2432 { pipe }
2433 { 0 }
2434 \ExplSyntaxOff

```

### 2.2.5.35 Strike-Through Renderer

The `\markdownRendererStrikeThrough` macro represents a strike-through span of text. The macro receives a single argument that corresponds to the striked-out span of text. This macro will only be produced, when the `strikeThrough` option is enabled.

```

2435 \def\markdownRendererStrikeThrough{%
2436 \markdownRendererStrikeThroughPrototype}%
2437 \ExplSyntaxOn
2438 \seq_gput_right:Nn
2439 \g_@@_renderers_seq
2440 { strikeThrough }
2441 \prop_gput:Nnn
2442 \g_@@_renderer_arities_prop
2443 { strikeThrough }

```

```
2444 { 1 }
2445 \ExplSyntaxOff
```

### 2.2.5.36 Subscript Renderer

The `\markdownRendererSubscript` macro represents a subscript span of text. The macro receives a single argument that corresponds to the subscript span of text. This macro will only be produced, when the `subscripts` option is enabled.

```
2446 \def\markdownRendererSubscript{%
2447 \markdownRendererSubscriptPrototype}%
2448 \ExplSyntaxOn
2449 \seq_gput_right:Nn
2450 \g_@@_renderers_seq
2451 { subscript }
2452 \prop_gput:Nnn
2453 \g_@@_renderer_arities_prop
2454 { subscript }
2455 { 1 }
```

### 2.2.5.37 Superscript Renderer

The `\markdownRendererSuperscript` macro represents a superscript span of text. The macro receives a single argument that corresponds to the superscript span of text. This macro will only be produced, when the `superscripts` option is enabled.

```
2456 \def\markdownRendererSuperscript{%
2457 \markdownRendererSuperscriptPrototype}%
2458 \ExplSyntaxOn
2459 \seq_gput_right:Nn
2460 \g_@@_renderers_seq
2461 { superscript }
2462 \prop_gput:Nnn
2463 \g_@@_renderer_arities_prop
2464 { superscript }
2465 { 1 }
2466 \ExplSyntaxOff
```

### 2.2.5.38 Table Attribute Context Renderers

The following macros are only produced, when the `tableCaptions` and `tableAttributes` options are enabled.

The `\markdownRendererTableAttributeContextBegin` and `\markdownRendererTableAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a table apply. The macros receive no arguments.

```
2467 \def\markdownRendererTableAttributeContextBegin{%
2468 \markdownRendererTableAttributeContextBeginPrototype}%
2469 \ExplSyntaxOn
```



```

2470 \seq_gput_right:Nn
2471 \g_@@_renderers_seq
2472 { tableAttributeContextBegin }
2473 \prop_gput:Nnn
2474 \g_@@_renderer_arities_prop
2475 { tableAttributeContextBegin }
2476 { 0 }
2477 \ExplSyntaxOff
2478 \def\markdownRendererTableAttributeContextEnd{%
2479 \markdownRendererTableAttributeContextEndPrototype}%
2480 \ExplSyntaxOn
2481 \seq_gput_right:Nn
2482 \g_@@_renderers_seq
2483 { tableAttributeContextEnd }
2484 \prop_gput:Nnn
2485 \g_@@_renderer_arities_prop
2486 { tableAttributeContextEnd }
2487 { 0 }
2488 \ExplSyntaxOff

```

### 2.2.5.39 Table Renderer

The `\markdownRendererTable` macro represents a table. This macro will only be produced, when the `pipeTables` option is enabled. The macro receives the parameters `{<caption>}{<number of rows>}{<number of columns>}` followed by `{<alignments>}` and then by `{<row>}` repeated `<number of rows>` times, where `<row>` is `{<column>}` repeated `<number of columns>` times, `<alignments>` is `<alignment>` repeated `<number of columns>` times, and `<alignment>` is one of the following:

- **d** – The corresponding column has an unspecified (default) alignment.
- **l** – The corresponding column is left-aligned.
- **c** – The corresponding column is centered.
- **r** – The corresponding column is right-aligned.

```

2489 \def\markdownRendererTable{%
2490 \markdownRendererTablePrototype}%
2491 \ExplSyntaxOn
2492 \seq_gput_right:Nn
2493 \g_@@_renderers_seq
2494 { table }
2495 \prop_gput:Nnn
2496 \g_@@_renderer_arities_prop
2497 { table }
2498 { 3 }
2499 \ExplSyntaxOff

```

#### 2.2.5.40 T<sub>E</sub>X Math Renderers

The `\markdownRendererInlineMath` and `\markdownRendererDisplayMath` macros represent inline and display T<sub>E</sub>X math. Both macros receive a single argument that corresponds to the T<sub>E</sub>X math content. These macros will only be produced, when the `texMathDollars`, `texMathSingleBackslash`, or `texMathDoubleBackslash` option are enabled.

```
2500 \def\markdownRendererInlineMath{%
2501 \markdownRendererInlineMathPrototype}%
2502 \ExplSyntaxOn
2503 \seq_gput_right:Nn
2504 \g_@@_renderers_seq
2505 { inlineMath }
2506 \prop_gput:Nnn
2507 \g_@@_renderer_arities_prop
2508 { inlineMath }
2509 { 1 }
2510 \ExplSyntaxOff
2511 \def\markdownRendererDisplayMath{%
2512 \markdownRendererDisplayMathPrototype}%
2513 \ExplSyntaxOn
2514 \seq_gput_right:Nn
2515 \g_@@_renderers_seq
2516 { displayMath }
2517 \prop_gput:Nnn
2518 \g_@@_renderer_arities_prop
2519 { displayMath }
2520 { 1 }
2521 \ExplSyntaxOff
```

#### 2.2.5.41 Thematic Break Renderer

The `\markdownRendererThematicBreak` macro represents a thematic break. The macro receives no arguments.

```
2522 \def\markdownRendererThematicBreak{%
2523 \markdownRendererThematicBreakPrototype}%
2524 \ExplSyntaxOn
2525 \seq_gput_right:Nn
2526 \g_@@_renderers_seq
2527 { thematicBreak }
2528 \prop_gput:Nnn
2529 \g_@@_renderer_arities_prop
2530 { thematicBreak }
2531 { 0 }
2532 \ExplSyntaxOff
```

#### 2.2.5.42 Tickbox Renderers

The macros named `\markdownRendererTickedBox`, `\markdownRendererHalfTickedBox`, and `\markdownRendererUntickedBox` represent ticked and unticked boxes, respectively. These macros will either be produced, when the `taskLists` option is enabled, or when the Ballot Box with X (☒, U+2612), Hourglass (⏏, U+231B) or Ballot Box (☐, U+2610) Unicode characters are encountered in the markdown input, respectively.

```

2533 \def\markdownRendererTickedBox{%
2534 \markdownRendererTickedBoxPrototype}%
2535 \ExplSyntaxOn
2536 \seq_gput_right:Nn
2537 \g_@@_renderers_seq
2538 { tickedBox }
2539 \prop_gput:Nnn
2540 \g_@@_renderer_arities_prop
2541 { tickedBox }
2542 { 0 }
2543 \ExplSyntaxOff
2544 \def\markdownRendererHalfTickedBox{%
2545 \markdownRendererHalfTickedBoxPrototype}%
2546 \ExplSyntaxOn
2547 \seq_gput_right:Nn
2548 \g_@@_renderers_seq
2549 { halfTickedBox }
2550 \prop_gput:Nnn
2551 \g_@@_renderer_arities_prop
2552 { halfTickedBox }
2553 { 0 }
2554 \ExplSyntaxOff
2555 \def\markdownRendererUntickedBox{%
2556 \markdownRendererUntickedBoxPrototype}%
2557 \ExplSyntaxOn
2558 \seq_gput_right:Nn
2559 \g_@@_renderers_seq
2560 { untickedBox }
2561 \prop_gput:Nnn
2562 \g_@@_renderer_arities_prop
2563 { untickedBox }
2564 { 0 }
2565 \ExplSyntaxOff

```

### 2.2.5.43 YAML Metadata Renderers

The `\markdownRendererJekyllDataBegin` macro represents the beginning of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

2566 \def\markdownRendererJekyllDataBegin{%
2567 \markdownRendererJekyllDataBeginPrototype}%

```

```

2568 \ExplSyntaxOn
2569 \seq_gput_right:Nn
2570 \g_@@_renderers_seq
2571 { jekyllDataBegin }
2572 \prop_gput:Nnn
2573 \g_@@_renderer_arities_prop
2574 { jekyllDataBegin }
2575 { 0 }
2576 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataEnd` macro represents the end of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

2577 \def\markdownRendererJekyllDataEnd{%
2578 \markdownRendererJekyllDataEndPrototype}%
2579 \ExplSyntaxOn
2580 \seq_gput_right:Nn
2581 \g_@@_renderers_seq
2582 { jekyllDataEnd }
2583 \prop_gput:Nnn
2584 \g_@@_renderer_arities_prop
2585 { jekyllDataEnd }
2586 { 0 }
2587 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the mapping.

```

2588 \def\markdownRendererJekyllDataMappingBegin{%
2589 \markdownRendererJekyllDataMappingBeginPrototype}%
2590 \ExplSyntaxOn
2591 \seq_gput_right:Nn
2592 \g_@@_renderers_seq
2593 { jekyllDataMappingBegin }
2594 \prop_gput:Nnn
2595 \g_@@_renderer_arities_prop
2596 { jekyllDataMappingBegin }
2597 { 2 }
2598 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingEnd` macro represents the end of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

2599 \def\markdownRendererJekyllDataMappingEnd{%

```

```

2600 \markdownRendererJekyllDataMappingEndPrototype}%
2601 \ExplSyntaxOn
2602 \seq_gput_right:Nn
2603 \g_@@_renderers_seq
2604 { jekyllDataMappingEnd }
2605 \prop_gput:Nnn
2606 \g_@@_renderer_arities_prop
2607 { jekyllDataMappingEnd }
2608 { 0 }
2609 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceBegin` macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the sequence.

```

2610 \def\markdownRendererJekyllDataSequenceBegin{%
2611 \markdownRendererJekyllDataSequenceBeginPrototype}%
2612 \ExplSyntaxOn
2613 \seq_gput_right:Nn
2614 \g_@@_renderers_seq
2615 { jekyllDataSequenceBegin }
2616 \prop_gput:Nnn
2617 \g_@@_renderer_arities_prop
2618 { jekyllDataSequenceBegin }
2619 { 2 }
2620 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceEnd` macro represents the end of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

2621 \def\markdownRendererJekyllDataSequenceEnd{%
2622 \markdownRendererJekyllDataSequenceEndPrototype}%
2623 \ExplSyntaxOn
2624 \seq_gput_right:Nn
2625 \g_@@_renderers_seq
2626 { jekyllDataSequenceEnd }
2627 \prop_gput:Nnn
2628 \g_@@_renderer_arities_prop
2629 { jekyllDataSequenceEnd }
2630 { 0 }
2631 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataBoolean` macro represents a boolean scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent

structure, and the scalar value, both cast to a string following YAML serialization rules.

```
2632 \def\markdownRendererJekyllDataBoolean{%
2633 \markdownRendererJekyllDataBooleanPrototype}%
2634 \ExplSyntaxOn
2635 \seq_gput_right:Nn
2636 \g_@@_renderers_seq
2637 { jekyllDataBoolean }
2638 \prop_gput:Nnn
2639 \g_@@_renderer_arities_prop
2640 { jekyllDataBoolean }
2641 { 2 }
2642 \ExplSyntaxOff
```

The `\markdownRendererJekyllDataNumber` macro represents a numeric scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```
2643 \def\markdownRendererJekyllDataNumber{%
2644 \markdownRendererJekyllDataNumberPrototype}%
2645 \ExplSyntaxOn
2646 \seq_gput_right:Nn
2647 \g_@@_renderers_seq
2648 { jekyllDataNumber }
2649 \prop_gput:Nnn
2650 \g_@@_renderer_arities_prop
2651 { jekyllDataNumber }
2652 { 2 }
2653 \ExplSyntaxOff
```

The `\markdownRendererJekyllDataString` macro represents a string scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

```
2654 \def\markdownRendererJekyllDataString{%
2655 \markdownRendererJekyllDataStringPrototype}%
2656 \ExplSyntaxOn
2657 \seq_gput_right:Nn
2658 \g_@@_renderers_seq
2659 { jekyllDataString }
2660 \prop_gput:Nnn
2661 \g_@@_renderer_arities_prop
2662 { jekyllDataString }
2663 { 2 }
2664 \ExplSyntaxOff
```

The `\markdownRendererJekyllDataEmpty` macro represents an empty scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives one argument: the scalar key in the parent structure, cast to a string following YAML serialization rules.

See also Section 2.2.6.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```

2665 \def\markdownRendererJekyllDataEmpty{%
2666 \markdownRendererJekyllDataEmptyPrototype}%
2667 \ExplSyntaxOn
2668 \seq_gput_right:Nn
2669 \g_@@_renderers_seq
2670 { jekyllDataEmpty }
2671 \prop_gput:Nnn
2672 \g_@@_renderer_arities_prop
2673 { jekyllDataEmpty }
2674 { 1 }
2675 \ExplSyntaxOff

```

#### 2.2.5.44 Generating Plain T<sub>E</sub>X Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain T<sub>E</sub>X macros for token renderers. Furthermore, the `\markdownSetup` macro also accepts the `renderers` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer macros and the values are new definitions of these token renderers.

```

2676 \ExplSyntaxOn
2677 \cs_new:Nn \@@_define_renderers:
2678 {
2679 \seq_map_function:NN
2680 \g_@@_renderers_seq
2681 \@@_define_renderer:n
2682 }
2683 \cs_new:Nn \@@_define_renderer:n
2684 {
2685 \@@_renderer_tl_to_csname:nN
2686 { #1 }
2687 \l_tmpa_tl
2688 \prop_get:NnN
2689 \g_@@_renderer_arities_prop
2690 { #1 }
2691 \l_tmpb_tl
2692 \@@_define_renderer:ncV
2693 { #1 }
2694 { \l_tmpa_tl }
2695 \l_tmpb_tl
2696 }

```

```

2697 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2698 {
2699 \tl_set:Nn
2700 \l_tmpa_tl
2701 { \str_uppercase:n { #1 } }
2702 \tl_set:Nx
2703 #2
2704 {
2705 markdownRenderer
2706 \tl_head:f { \l_tmpa_tl }
2707 \tl_tail:n { #1 }
2708 }
2709 }
2710 \tl_new:N
2711 \l_@@_renderer_definition_tl
2712 \cs_new:Nn \@@_define_renderer:nNn
2713 {
2714 \keys_define:nn
2715 { markdown/options/renderers }
2716 {
2717 #1 .code:n = {
2718 \tl_set:Nn
2719 \l_@@_renderer_definition_tl
2720 { ##1 }
2721 \regex_replace_all:nnN
2722 { \cP\#0 }
2723 { #1 }
2724 \l_@@_renderer_definition_tl
2725 \cs_generate_from_arg_count:NNnV
2726 #2
2727 \cs_set:Npn
2728 { #3 }
2729 \l_@@_renderer_definition_tl
2730 },
2731 }
2732 }
2733 \cs_generate_variant:Nn
2734 \@@_define_renderer:nNn
2735 { ncV }
2736 \cs_generate_variant:Nn
2737 \cs_generate_from_arg_count:NNnn
2738 { NNnV }
2739 \keys_define:nn
2740 { markdown/options }
2741 {
2742 renderers .code:n = {
2743 \keys_set:nn

```



```

2744 { markdown/options/renderers }
2745 { #1 }
2746 },
2747 }
2748 \ExplSyntaxOff

```

The following example code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` token renderer macros.

```

\markdownSetup{
 renderers = {
 link = {#4}, % Render links as the link title.
 emphasis = {{\it #1}}, % Render emphasized text using italics.
 }
}

```

In addition to exact token renderer names, we also support wildcards and enumerations that match multiple token renderer names:

```

\markdownSetup{
 renderers = {
 heading* = {{\bf #1}}, % Render headings using the bold face.
 jekyllData(String|Number) = { % Render YAML string and numbers
 {\it #2}% % using italics.
 },
 }
}

```

Wildcards and enumerations can be combined:

```

\markdownSetup{
 renderers = {
 *lItem(|End) = {"}, % Quote ordered/bullet list items.
 }
}

```

To determine the current token renderer, you can use the parameter `#0`:

```

\markdownSetup{
 renderers = {
 heading* = {#0: #1}, % Render headings as the renderer name
 } % followed by the heading text.
}

```

```

2749 \ExplSyntaxOn
2750 \prop_new:N
2751 \g_@@_glob_cache_prop
2752 \tl_new:N
2753 \l_@@_current_glob_tl
2754 \cs_new:Nn
2755 \@@_glob_seq:nnN
2756 {
2757 \tl_set:Nn
2758 \l_@@_current_glob_tl
2759 { ^ #1 $ }
2760 \prop_get:NeNTF
2761 \g_@@_glob_cache_prop
2762 { #2 / \l_@@_current_glob_tl }
2763 \l_tmpa_clist
2764 {
2765 \seq_set_from_clist:NN
2766 #3
2767 \l_tmpa_clist
2768 }
2769 {
2770 \seq_clear:N
2771 #3
2772 \regex_replace_all:nnN
2773 { * }
2774 { .* }
2775 \l_@@_current_glob_tl
2776 \regex_set:NV
2777 \l_tmpa_regex
2778 \l_@@_current_glob_tl
2779 \seq_map_inline:cn
2780 { #2 }
2781 {
2782 \regex_match:NnT
2783 \l_tmpa_regex
2784 { ##1 }
2785 {
2786 \seq_put_right:Nn
2787 #3
2788 { ##1 }
2789 }
2790 }
2791 \clist_set_from_seq:NN
2792 \l_tmpa_clist
2793 #3
2794 \prop_gput:NeV
2795 \g_@@_glob_cache_prop

```

```

2796 { #2 / \l_@@_current_glob_tl }
2797 \l_tmpa_clist
2798 }
2799 }
2800 % TODO: Remove in TeX Live 2023.
2801 \prg_generate_conditional_variant:Nnn
2802 \prop_get:NnN
2803 { NeN }
2804 { TF }
2805 \cs_generate_variant:Nn
2806 \regex_set:Nn
2807 { NV }
2808 \cs_generate_variant:Nn
2809 \prop_gput:Nnn
2810 { NeV }
2811 \seq_new:N
2812 \l_@@_renderer_glob_results_seq
2813 \keys_define:nn
2814 { markdown/options/renderers }
2815 {
2816 unknown .code:n = {
2817 \@@_glob_seq:VnN
2818 \l_keys_key_str
2819 { g_@@_renderers_seq }
2820 \l_@@_renderer_glob_results_seq
2821 \seq_if_empty:NTF
2822 \l_@@_renderer_glob_results_seq
2823 {
2824 \msg_error:nnV
2825 { markdown }
2826 { undefined-renderer }
2827 \l_keys_key_str
2828 }
2829 }
2830 \tl_set:Nn
2831 \l_@@_renderer_definition_tl
2832 { #1 }
2833 \seq_map_inline:Nn
2834 \l_@@_renderer_glob_results_seq
2835 {
2836 \@@_renderer_tl_to_csname:nN
2837 { ##1 }
2838 \l_tmpa_tl
2839 \prop_get:NnN
2840 \g_@@_renderer_arities_prop
2841 { ##1 }
2842 \l_tmpb_tl

```

```

2843 \int_set:Nn
2844 \l_tmpa_int
2845 \l_tmpb_tl
2846 \tl_set:NV
2847 \l_tmpb_tl
2848 \l_@@_renderer_definition_tl
2849 \regex_replace_all:nnN
2850 { \cP\#0 }
2851 { ##1 }
2852 \l_tmpb_tl
2853 \cs_generate_from_arg_count:cNVV
2854 { \l_tmpa_tl }
2855 \cs_set:Npn
2856 \l_tmpa_int
2857 \l_tmpb_tl
2858 }
2859 }
2860 },
2861 }
2862 \msg_new:nnn
2863 { markdown }
2864 { undefined-renderer }
2865 {
2866 Renderer~#1~is~undefined.
2867 }
2868 \cs_generate_variant:Nn
2869 \@@_glob_seq:nnN
2870 { VnN }
2871 \cs_generate_variant:Nn
2872 \cs_generate_from_arg_count:NNnn
2873 { cNVV }
2874 \cs_generate_variant:Nn
2875 \msg_error:nnn
2876 { nnV }

```

If plain  $\text{T}_{\text{E}}\text{X}$  is the top layer, we use the `\@@_define_renderers:` macro to define plain  $\text{T}_{\text{E}}\text{X}$  token renderer macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

2877 \str_if_eq:VVT
2878 \c_@@_top_layer_tl
2879 \c_@@_option_layer_plain_tex_tl
2880 {
2881 \@@_define_renderers:
2882 }
2883 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the `LATEX3` kernel.

```
2884 \ExplSyntaxOn
2885 \keys_define:nn
2886 { markdown/jekyllData }
2887 { }
2888 \ExplSyntaxOff
```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key-values without using the `expl3` language.

```
2889 \ExplSyntaxOn
2890 \@@_with_various_cases:nn
2891 { jekyllDataRenderers }
2892 {
2893 \keys_define:nn
2894 { markdown/options }
2895 {
2896 #1 .code:n = {
2897 \tl_set:Nn
2898 \l_tmpa_tl
2899 { ##1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```
2900 \tl_replace_all:NnV
2901 \l_tmpa_tl
2902 { / }
2903 \c_backslash_str
2904 \keys_set:nV
2905 { markdown/options/jekyll-data-renderers }
2906 \l_tmpa_tl
2907 },
2908 }
2909 }
2910 \keys_define:nn
2911 { markdown/options/jekyll-data-renderers }
2912 {
2913 unknown .code:n = {
2914 \tl_set_eq:NN
2915 \l_tmpa_tl
```

```

2916 \l_keys_key_str
2917 \tl_replace_all:NVn
2918 \l_tmpa_tl
2919 \c_backslash_str
2920 { / }
2921 \tl_put_right:Nn
2922 \l_tmpa_tl
2923 {
2924 .code:n = { #1 }
2925 }
2926 \keys_define:nV
2927 { markdown/jekyllData }
2928 \l_tmpa_tl
2929 }
2930 }
2931 \cs_generate_variant:Nn
2932 \keys_define:nn
2933 { nV }
2934 \ExplSyntaxOff

```

### 2.2.6.2 Generating Plain TeX Token Renderer Prototype Macros and Key-Values

We define the command `\@@_define_renderer_prototypes:` that defines plain TeX macros for token renderer prototypes. Furthermore, the `\markdownSetup` macro also accepts the `rendererPrototype` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer prototype macros and the values are new definitions of these token renderer prototypes.

```

2935 \ExplSyntaxOn
2936 \cs_new:Nn \@@_define_renderer_prototypes:
2937 {
2938 \seq_map_function:NN
2939 \g_@@_renderers_seq
2940 \@@_define_renderer_prototype:n
2941 }
2942 \cs_new:Nn \@@_define_renderer_prototype:n
2943 {
2944 \@@_renderer_prototype_tl_to_csname:nN
2945 { #1 }
2946 \l_tmpa_tl
2947 \prop_get:NnN
2948 \g_@@_renderer_arities_prop
2949 { #1 }
2950 \l_tmpb_tl
2951 \@@_define_renderer_prototype:ncV
2952 { #1 }
2953 { \l_tmpa_tl }

```

```

2954 \l_tmpb_tl
2955 }
2956 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
2957 {
2958 \tl_set:Nn
2959 \l_tmpa_tl
2960 { \str_uppercase:n { #1 } }
2961 \tl_set:Nx
2962 #2
2963 {
2964 markdownRenderer
2965 \tl_head:f { \l_tmpa_tl }
2966 \tl_tail:n { #1 }
2967 Prototype
2968 }
2969 }
2970 \tl_new:N
2971 \l_@@_renderer_prototype_definition_tl
2972 \cs_new:Nn \@@_define_renderer_prototype:nNn
2973 {
2974 \keys_define:nn
2975 { markdown/options/renderer-prototypes }
2976 {
2977 #1 .code:n = {
2978 \tl_set:Nn
2979 \l_@@_renderer_prototype_definition_tl
2980 { ##1 }
2981 \regex_replace_all:nnN
2982 { \cP\#0 }
2983 { #1 }
2984 \l_@@_renderer_prototype_definition_tl
2985 \cs_generate_from_arg_count:NNnV
2986 #2
2987 \cs_set:Npn
2988 { #3 }
2989 \l_@@_renderer_prototype_definition_tl
2990 },
2991 }

```

Unless the token renderer prototype macro has already been defined, we provide an empty definition.

```

2992 \cs_if_free:NT
2993 #2
2994 {
2995 \cs_generate_from_arg_count:NNnn
2996 #2
2997 \cs_set:Npn

```

```

2998 { #3 }
2999 { }
3000 }
3001 }
3002 \cs_generate_variant:Nn
3003 \@@_define_renderer_prototype:nNn
3004 { ncV }
3005 \ExplSyntaxOff

```

The following example code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` token renderer prototype macros.

```

\markdownSetup{
 rendererPrototypes = {
 image = {\pdfximage{#2}}, % Embed PDF images in the document.
 codeSpan = {\tt #1}, % Render inline code using monospace.
 }
}

```

In addition to exact token renderer names, we also support wildcards and enumerations that match multiple token renderer prototype names:

```

\markdownSetup{
 rendererPrototypes = {
 heading* = {\bf #1}, % Render headings using the bold face.
 jekyllData(String|Number) = { % Render YAML string and numbers
 {\it #2}% % using italics.
 },
 }
}

```

Wildcards and enumerations can be combined:

```

\markdownSetup{
 rendererPrototypes = {
 *lItem(|End) = {"}, % Quote ordered/bullet list items.
 }
}

```

To determine the current token renderer prototype, you can use the parameter `#0`:



```

\markdownSetup{
 rendererPrototypes = {
 heading* = {#0: #1}, % Render headings as the renderer prototype
 }
 % name followed by the heading text.
}

```

```

3006 \ExplSyntaxOn
3007 \seq_new:N
3008 \l_@@_renderer_prototype_glob_results_seq
3009 \keys_define:nn
3010 { markdown/options/renderer-prototypes }
3011 {
3012 unknown .code:n = {
3013 \@@_glob_seq:VnN
3014 \l_keys_key_str
3015 { g_@@_renderers_seq }
3016 \l_@@_renderer_prototype_glob_results_seq
3017 \seq_if_empty:NTF
3018 \l_@@_renderer_prototype_glob_results_seq
3019 {
3020 \msg_error:nnV
3021 { markdown }
3022 { undefined-renderer-prototype }
3023 \l_keys_key_str
3024 }
3025 {
3026 \tl_set:Nn
3027 \l_@@_renderer_prototype_definition_tl
3028 { #1 }
3029 \seq_map_inline:Nn
3030 \l_@@_renderer_prototype_glob_results_seq
3031 {
3032 \@@_renderer_prototype_tl_to_csname:nN
3033 { ##1 }
3034 \l_tmpa_tl
3035 \prop_get:NnN
3036 \g_@@_renderer_arities_prop
3037 { ##1 }
3038 \l_tmpb_tl
3039 \int_set:Nn
3040 \l_tmpa_int
3041 \l_tmpb_tl
3042 \tl_set:NV
3043 \l_tmpb_tl
3044 \l_@@_renderer_prototype_definition_tl

```

```

3045 \regex_replace_all:nnN
3046 { \cP\#0 }
3047 { ##1 }
3048 \l_tmpb_tl
3049 \cs_generate_from_arg_count:cN\V
3050 { \l_tmpa_tl }
3051 \cs_set:Npn
3052 \l_tmpa_int
3053 \l_tmpb_tl
3054 }
3055 }
3056 },
3057 }
3058 \msg_new:nnn
3059 { markdown }
3060 { undefined-renderer-prototype }
3061 {
3062 Renderer~prototype~#1~is~undefined.
3063 }
3064 \@@_with_various_cases:nn
3065 { rendererPrototypes }
3066 {
3067 \keys_define:nn
3068 { markdown/options }
3069 {
3070 #1 .code:n = {
3071 \keys_set:nn
3072 { markdown/options/renderer-prototypes }
3073 { ##1 }
3074 },
3075 }
3076 }

```

If plain  $\text{T}_{\text{E}}\text{X}$  is the top layer, we use the `\@@_define_renderer_prototypes:` macro to define plain  $\text{T}_{\text{E}}\text{X}$  token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3077 \str_if_eq:VVT
3078 \c_@@_top_layer_tl
3079 \c_@@_option_layer_plain_tex_tl
3080 {
3081 \@@_define_renderer_prototypes:
3082 }
3083 \ExplSyntaxOff

```

## 2.2.7 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros perform logging for the Markdown package. Their first argument specifies the text of the info, warning, or error message. The `\markdownError` macro receives a second argument that provides a help text. You may redefine these macros to redirect and process the info, warning, and error messages.

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros have been deprecated and will be removed in the next major version of the Markdown package.

## 2.2.8 Miscellanea

The `\markdownMakeOther` macro is used by the package, when a  $\TeX$  engine that does not support direct Lua access is starting to buffer a text. The plain  $\TeX$  implementation changes the category code of plain  $\TeX$  special characters to *other*, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
3084 \let\markdownMakeOther\relax
```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain  $\TeX$  special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
3085 \let\markdownReadAndConvert\relax
3086 \begingroup
```

Locally swap the category code of the backslash symbol (`\`) with the pipe symbol (`|`). This is required in order that all the special symbols in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```
3087 \catcode`\|=0\catcode`\=12%
3088 |gdef|markdownBegin{%
3089 |markdownReadAndConvert{\markdownEnd}%
3090 {|\markdownEnd}}%
3091 |endgroup
```

The macro is exposed in the interface, so that users can create their own markdown environments. Due to the way the arguments are passed to Lua, the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol).

The `code` key, which can be used to immediately expand and execute code.

```
3092 \ExplSyntaxOn
3093 \keys_define:nn
3094 { markdown/options }
3095 {
3096 code .code:n = { #1 },
```

```

3097 }
3098 \ExplSyntaxOff

```

This can be especially useful in snippets.

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

To determine whether L<sup>A</sup>T<sub>E</sub>X is the top layer or if there are other layers above L<sup>A</sup>T<sub>E</sub>X, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that L<sup>A</sup>T<sub>E</sub>X is the top layer.

```

3099 \ExplSyntaxOn
3100 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3101 \cs_generate_variant:Nn
3102 \tl_const:Nn
3103 { NV }
3104 \tl_if_exist:NF
3105 \c_@@_top_layer_tl
3106 {
3107 \tl_const:NV
3108 \c_@@_top_layer_tl
3109 \c_@@_option_layer_latex_tl
3110 }
3111 \ExplSyntaxOff
3112 \input markdown/markdown

```

The L<sup>A</sup>T<sub>E</sub>X interface is implemented by the `markdown.sty` file, which can be loaded from the L<sup>A</sup>T<sub>E</sub>X document preamble as follows:

```
\usepackage[<options>]{markdown}
```

where `<options>` are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2). Note that `<options>` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.44) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single L<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. Both L<sup>A</sup>T<sub>E</sub>X environments accept L<sup>A</sup>T<sub>E</sub>X interface options (see section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown*` environment has been deprecated and will be removed in the next major version of the Markdown package.

```
3113 \newenvironment{markdown}\relax\relax
3114 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\markdownEnd` macro to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `markdown` and `markdown*` environments:

<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown}[smartEllipses] _Hello_ **world** ... \end{markdown} % ... \end{document}</pre>	<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown*}{smartEllipses} _Hello_ **world** ... \end{markdown*} % ... \end{document}</pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `\markdownInput` macro:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[smartEllipses]{hello.md}
\end{document}
```

## 2.3.2 Options

The  $\LaTeX$  options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  if the  $= \langle value \rangle$  part has been omitted.

$\LaTeX$  options map directly to the options recognized by the plain  $\TeX$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\TeX$  interface (see Sections 2.2.5 and 2.2.6).

The  $\LaTeX$  options may be specified when loading the  $\LaTeX$  package, when using the `markdown*`  $\LaTeX$  environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

### 2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [9, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\TeX$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

```
\documentclass[frozenscache]{article}
\usepackage{markdown,minted}
\begin{document}
\end{document}
```

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing  $\LaTeX$  document sources for distribution.

```
3115 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3116 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

### 2.3.2.2 Generating Plain $\TeX$ Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

If  $\LaTeX$  is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to define plain  $\TeX$  option, token renderer, and token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
3117 \ExplSyntaxOn
3118 \str_if_eq:VVT
3119 \c_@@_top_layer_tl
3120 \c_@@_option_layer_latex_tl
```

```

3121 {
3122 \@@_define_option_commands_and_keyvals:
3123 \@@_define_renderers:
3124 \@@_define_renderer_prototypes:
3125 }
3126 \ExplSyntaxOff

```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of plain T<sub>E</sub>X interface options `hybrid`, `smartEllipses`, and `cacheDir`.

```

\markdownSetup{
 hybrid,
 smartEllipses,
 cacheDir = /tmp,
}

```

### 2.3.3 Themes

In Section 2.2.3, we described the concept of themes. In L<sup>A</sup>T<sub>E</sub>X, we expand on the concept of themes by allowing a theme to be a full-blown L<sup>A</sup>T<sub>E</sub>X package. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a L<sup>A</sup>T<sub>E</sub>X package named `markdowntheme<munged theme name>.sty` if it exists and a T<sub>E</sub>X document named `markdowntheme<munged theme name>.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex theme file` or the L<sup>A</sup>T<sub>E</sub>X-specific `.sty theme file` allows developers to have a single *theme file*, when the theme is small or the difference between T<sub>E</sub>X formats is unimportant, and scale up to separate theme files native to different T<sub>E</sub>X formats for large multi-format themes, where different code is needed for different T<sub>E</sub>X formats. To enable code reuse, developers can load the `.tex theme file` from the `.sty theme file` using the `\markdownLoadPlainTeXTheme` macro.

If the L<sup>A</sup>T<sub>E</sub>X option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown L<sup>A</sup>T<sub>E</sub>X package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` L<sup>A</sup>T<sub>E</sub>X package, and finally the `markdownthemewitiko_dot.sty` L<sup>A</sup>T<sub>E</sub>X package:

```

\usepackage[
 import=witiko/beamer/MU,
 import=witiko/dot,
]{markdown}

```

```
3127 \newif\ifmarkdownLaTeXLoaded
3128 \markdownLaTeXLoadedfalse
```

Due to limitations of L<sup>A</sup>T<sub>E</sub>X, themes may not be loaded after the beginning of a L<sup>A</sup>T<sub>E</sub>X document.

Built-in L<sup>A</sup>T<sub>E</sub>X themes provided with the Markdown package include:

**witiko/dot** A theme that typesets fenced code blocks with the `dot ...` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

```
\documentclass{article}
\usepackage[import=witiko/dot]{markdown}
\setkeys{Gin}{
 width = \columnwidth,
 height = 0.65\paperheight,
 keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

  latex -> pmml;
  latex -> cmml;
  pmml -> slt;
  cmml -> opt;
  cmml -> prefix;
  cmml -> infix;
  pmml -> mterms [style=dashed];
  cmml -> mterms;

  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];
  infix [label = "Infix"];
  mterms [label = "M-Terms"];
}
```
```

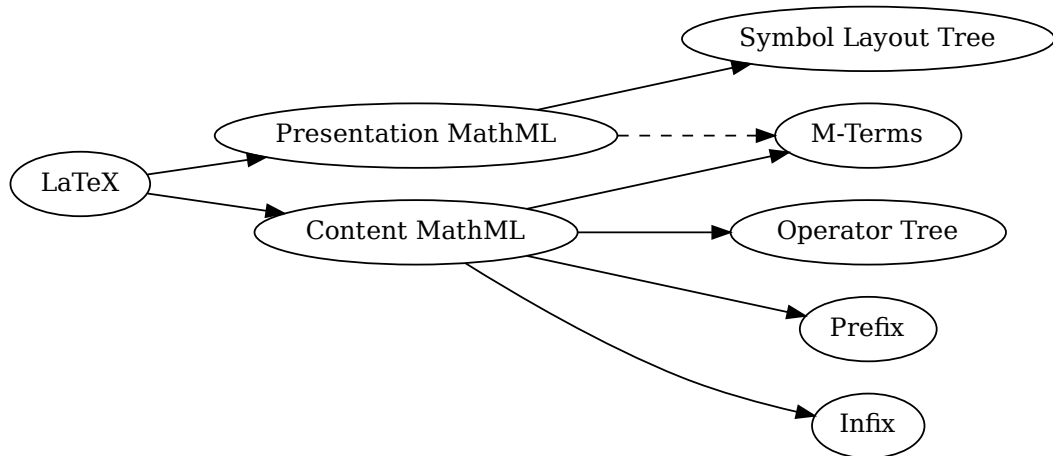


```

\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in Figure 4.



**Figure 4: Various formats of mathematical formulae**

The theme requires a Unix-like operating system with GNU Diffutils and Graphviz installed. The theme also requires shell access unless the `frozenCache` plain  $\TeX$  option is enabled.

```

3129 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%

```

**witiko/graphicx/http** A theme that adds support for downloading images whose URL has the http or https protocol.

```

\documentclass{article}
\usepackage[import=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}
! [img] (https://github.com/witiko/markdown/raw/main/markdown.png
 "The banner of the Markdown package")
\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile  $\LaTeX$  package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The

```

\documentclass{book}
\usepackage{markdown}
\markdownSetup{pipeTables,tableCaptions}
\begin{document}
\begin{markdown}
Introduction
=====
Section
Subsection
Hello *Markdown*!

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

: Table
\end{markdown}
\end{document}

```



# Chapter 1

## Introduction

1.1 Section  
1.1.1 Subsection  
Hello *Markdown!*

| Right | Left | Default | Center |
|-------|------|---------|--------|
| 12    | 12   | 12      | 12     |
| 123   | 123  | 123     | 123    |
| 1     | 1    | 1       | 1      |

Table 1.1: Table

**Figure 5: The banner of the Markdown package**

theme also requires shell access unless the `frozenCache` plain  $\TeX$  option is enabled.

```
3130 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

**witiko/markdown/defaults** A  $\LaTeX$  theme with the default definitions of token renderer prototypes for plain  $\TeX$ . This theme is loaded automatically together with the package and explicitly loading it has no effect.

```

3131 \AtEndOfPackage{
3132 \markdownLaTeXLoadedtrue

At the end of the \LaTeX module, we load the witiko/markdown/defaults \LaTeX theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option noDefaults has been enabled (see Section 2.2.2.3).

3133 \markdownIfOption{noDefaults}{-}{
3134 \markdownSetup{theme=witiko/markdown/defaults}
3135 }
3136 }

3137 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%

```

Please, see Section 3.3.4 for implementation details of the built-in  $\LaTeX$  themes.

## 2.4 ConT<sub>E</sub>Xt Interface

To determine whether ConT<sub>E</sub>Xt is the top layer or if there are other layers above ConT<sub>E</sub>Xt, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that ConT<sub>E</sub>Xt is the top layer.

```
3138 \ExplSyntaxOn
3139 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3140 \cs_generate_variant:Nn
3141 \tl_const:Nn
3142 { NV }
3143 \tl_if_exist:NF
3144 \c_@@_top_layer_tl
3145 {
3146 \tl_const:NV
3147 \c_@@_top_layer_tl
3148 \c_@@_option_layer_context_tl
3149 }
3150 \ExplSyntaxOff
```

The ConT<sub>E</sub>Xt interface provides a start-stop macro pair for the typesetting of markdown input from within ConT<sub>E</sub>Xt and facilities for setting Lua, plain T<sub>E</sub>X, and ConT<sub>E</sub>Xt options used during the conversion from markdown to plain T<sub>E</sub>X. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```
3151 \writestatus{loading}{ConTeXt User Module / markdown}%
3152 \startmodule[markdown]
3153 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3154 \do#\do\^\do_do\%do\~}%
3155 \input markdown/markdown
```

The ConT<sub>E</sub>Xt interface is implemented by the `t-markdown.tex` ConT<sub>E</sub>Xt module file that can be loaded as follows:

```
\usemodule[t][markdown]
```

It is expected that the special plain T<sub>E</sub>X characters have the expected category codes, when `\inputting` the file.

### 2.4.1 Typesetting Markdown

The interface exposes the `\startmarkdown` and `\stopmarkdown` macro pair for the typesetting of a markdown document fragment, and defines the `\inputmarkdown` macro.

```
3156 \let\startmarkdown\relax
3157 \let\stopmarkdown\relax
3158 \let\inputmarkdown\relax
```

You may prepend your own code to the `\startmarkdown` macro and redefine the `\stopmarkdown` macro to produce special effects before and after the markdown block.

Note that the `\startmarkdown` and `\stopmarkdown` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain `TEX` interface.

The following example `ConTEXt` code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```
\usemodule[t] [markdown]
\starttext
\startmarkdown
Hello world ...
\stopmarkdown
\stoptext
```

The `\inputmarkdown` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain `TEX`. Unlike the `\markdownInput` macro provided by the plain `TEX` interface, this macro also accepts `ConTEXt` interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example `LATEX` code showcases the usage of the `\markdownInput` macro:

```
\usemodule[t] [markdown]
\starttext
\inputmarkdown[smartEllipses]{hello.md}
\stoptext
```

## 2.4.2 Options

The `ConTEXt` options are represented by a comma-delimited list of `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` (or, equivalently, `<key>=yes`) if the `=<value>` part has been omitted.

`ConTEXt` options map directly to the options recognized by the plain `TEX` interface (see Section 2.2.2).

The `ConTEXt` options may be specified when using the `\inputmarkdown` macro (see Section 2.4), via the `\markdownSetup` macro, or via the `\setupmarkdown[#1]` macro, which is an alias for `\markdownSetup{#1}`.

3159 `\ExplSyntaxOn`

```

3160 \cs_new:Npn
3161 \setupmarkdown
3162 [#1]
3163 {
3164 \@@_setup:n
3165 { #1 }
3166 }
3167 \ExplSyntaxOff

```

### 2.4.2.1 Generating Plain T<sub>E</sub>X Option Macros and Key-Values

Unlike plain T<sub>E</sub>X, we also accept caseless variants of options in line with the style of ConT<sub>E</sub>Xt.

```

3168 \ExplSyntaxOn
3169 \cs_new:Nn \@@_caseless:N
3170 {
3171 \regex_replace_all:nnN
3172 { ([a-z])([A-Z]) }
3173 { \1 \c { str_lowercase:n } \cB{\ 2 \cE\} }
3174 #1
3175 \tl_set:Nx
3176 #1
3177 { #1 }
3178 }
3179 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

If ConT<sub>E</sub>Xt is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to define plain T<sub>E</sub>X option, token renderer, and token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3180 \str_if_eq:VVT
3181 \c_@@_top_layer_tl
3182 \c_@@_option_layer_context_tl
3183 {
3184 \@@_define_option_commands_and_keyvals:
3185 \@@_define_renderers:
3186 \@@_define_renderer_prototypes:
3187 }
3188 \ExplSyntaxOff

```

### 2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConT<sub>E</sub>Xt, we expand on the concept of themes by allowing a theme to be a full-blown ConT<sub>E</sub>Xt module. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load

a ConT<sub>E</sub>Xt module named `t-markdowntheme⟨munged theme name⟩.tex` if it exists and a T<sub>E</sub>X document named `markdowntheme⟨munged theme name⟩.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex theme file` or the ConT<sub>E</sub>Xt-specific `t-*.tex` theme file allows developers to have a single *theme file*, when the theme is small or the difference between T<sub>E</sub>X formats is unimportant, and scale up to separate theme files native to different T<sub>E</sub>X formats for large multi-format themes, where different code is needed for different T<sub>E</sub>X formats. To enable code reuse, developers can load the `.tex` theme file from the `t-*.tex` theme file using the `\markdownLoadPlainTeXTheme` macro.

For example, to load a theme named `witiko/tilde` in your document:

```
\usemodule[t][markdown]
\setupmarkdown[import=witiko/tilde]
```

Built-in ConT<sub>E</sub>Xt themes provided with the Markdown package include:

**witiko/markdown/defaults** A ConT<sub>E</sub>Xt theme with the default definitions of token renderer prototypes for plain T<sub>E</sub>X. This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
3189 \startmodule[markdownthemewitiko_markdown_defaults]
3190 \unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConT<sub>E</sub>Xt themes.

## 3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to T<sub>E</sub>X *token renderers* is performed by the Lua layer. The plain T<sub>E</sub>X layer provides default definitions for the token renderers. The L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt layers correct idiosyncrasies of the respective T<sub>E</sub>X formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain T<sub>E</sub>X, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module

and the remaining markdown reader and plain T<sub>E</sub>X writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3191 local upper, format, length =
3192 string.upper, string.format, string.len
3193 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3194 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3195 lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

### 3.1.1 Utility Functions

This section documents the utility functions used by the plain T<sub>E</sub>X writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
3196 local util = {}
```

The `util.err` method prints an error message `msg` and exits. If `exit_code` is provided, it specifies the exit code. Otherwise, the exit code will be 1.

```
3197 function util.err(msg, exit_code)
3198 io.stderr:write("markdown.lua: " .. msg .. "\n")
3199 os.exit(exit_code or 1)
3200 end
```

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```
3201 function util.cache(dir, string, salt, transform, suffix)
3202 local digest = md5.sumhexa(string .. (salt or ""))
3203 local name = util.pathname(dir, digest .. suffix)
3204 local file = io.open(name, "r")
3205 if file == nil then -- If no cache entry exists, then create a new one.
3206 file = assert(io.open(name, "w"),
3207 [[Could not open file]] .. name .. [[for writing]])
3208 local result = string
3209 if transform ~= nil then
3210 result = transform(result)
3211 end
3212 assert(file:write(result))
3213 assert(file:close())
3214 end
3215 return name
3216 end
```

The `util.cache_verbatim` method strips whitespaces from the end of `string` and calls `util.cache` with `dir`, `string`, no salt or transformations, and the `.verbatim` suffix.

```

3217 function util.cache_verbatim(dir, string)
3218 local name = util.cache(dir, string, nil, nil, ".verbatim")
3219 return name
3220 end

```

The `util.table_copy` method creates a shallow copy of a table `t` and its metatable.

```

3221 function util.table_copy(t)
3222 local u = { }
3223 for k, v in pairs(t) do u[k] = v end
3224 return setmetatable(u, getmetatable(t))
3225 end

```

The `util.encode_json_string` method encodes a string `s` in JSON.

```

3226 function util.encode_json_string(s)
3227 s = s:gsub([[\\]], [[\\]])
3228 s = s:gsub([[\"]], [[\"]])
3229 return [[\"]] .. s .. [[\"]]
3230 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimsky [10, Chapter 21].

```

3231 function util.expand_tabs_in_line(s, tabstop)
3232 local tab = tabstop or 4
3233 local corr = 0
3234 return (s:gsub(")\t", function(p)
3235 local sp = tab - (p - 1 + corr) % tab
3236 corr = corr - 1 + sp
3237 return string.rep(" ", sp)
3238 end))
3239 end

```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

```

3240 function util.walk(t, f)
3241 local typ = type(t)
3242 if typ == "string" then
3243 f(t)
3244 elseif typ == "table" then
3245 local i = 1
3246 local n
3247 n = t[i]
3248 while n do
3249 util.walk(n, f)
3250 i = i + 1

```



```

3251 n = t[i]
3252 end
3253 elseif typ == "function" then
3254 local ok, val = pcall(t)
3255 if ok then
3256 util.walk(val,f)
3257 end
3258 else
3259 f(tostring(t))
3260 end
3261 end

```

The `util.flatten` method flattens an array `ary` that does not contain cycles and returns the result.

```

3262 function util.flatten(ary)
3263 local new = {}
3264 for _,v in ipairs(ary) do
3265 if type(v) == "table" then
3266 for _,w in ipairs(util.flatten(v)) do
3267 new[#new + 1] = w
3268 end
3269 else
3270 new[#new + 1] = v
3271 end
3272 end
3273 return new
3274 end

```

The `util.rope_to_string` method converts a rope `rope` to a string and returns it. For the definition of a rope, see the definition of the `util.walk` method.

```

3275 function util.rope_to_string(rope)
3276 local buffer = {}
3277 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3278 return table.concat(buffer)
3279 end

```

The `util.rope_last` method retrieves the last item in a rope. For the definition of a rope, see the definition of the `util.walk` method.

```

3280 function util.rope_last(rope)
3281 if #rope == 0 then
3282 return nil
3283 else
3284 local l = rope[#rope]
3285 if type(l) == "table" then
3286 return util.rope_last(l)
3287 else
3288 return l
3289 end
3290 end

```

```

3290 end
3291 end

```

Given an array `ary` and a string `x`, the `util.intersperse` method returns an array `new`, such that `ary[i] == new[2*(i-1)+1]` and `new[2*i] == x` for all  $1 \leq i \leq \#ary$ .

```

3292 function util.intersperse(ary, x)
3293 local new = {}
3294 local l = #ary
3295 for i,v in ipairs(ary) do
3296 local n = #new
3297 new[n + 1] = v
3298 if i ~= l then
3299 new[n + 2] = x
3300 end
3301 end
3302 return new
3303 end

```

Given an array `ary` and a function `f`, the `util.map` method returns an array `new`, such that `new[i] == f(ary[i])` for all  $1 \leq i \leq \#ary$ .

```

3304 function util.map(ary, f)
3305 local new = {}
3306 for i,v in ipairs(ary) do
3307 new[i] = f(v)
3308 end
3309 return new
3310 end

```

Given a table `char_escapes` mapping escapable characters to escaped strings and optionally a table `string_escapes` mapping escapable strings to escaped strings, the `util.escaper` method returns an escaper function that escapes all occurrences of escapable strings and characters (in this order).

The method uses LPeg, which is faster than the Lua `string.gsub` built-in method.

```

3311 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

3312 local char_escapes_list = ""
3313 for i,_ in pairs(char_escapes) do
3314 char_escapes_list = char_escapes_list .. i
3315 end

```

Create an LPeg capture `escapable` that produces the escaped string corresponding to the matched escapable character.

```

3316 local escapable = S(char_escapes_list) / char_escapes

```

If `string_escapes` is provided, turn `escapable` into the

$$\sum_{(k,v) \in \text{string\_escapes}} P(k) / v + \text{escapable}$$

capture that replaces any occurrence of the string `k` with the string `v` for each  $(k, v) \in \text{string\_escapes}$ . Note that the pattern summation is not commutative and its operands are inspected in the summation order during the matching. As a corollary, the strings always take precedence over the characters.

```
3317 if string_escapes then
3318 for k,v in pairs(string_escapes) do
3319 escapable = P(k) / v + escapable
3320 end
3321 end
```

Create an LPeg capture `escape_string` that captures anything `escapable` does and matches any other unmatched characters.

```
3322 local escape_string = Cs((escapable + any)^0)
```

Return a function that matches the input string `s` against the `escape_string` capture.

```
3323 return function(s)
3324 return lpeg.match(escape_string, s)
3325 end
3326 end
```

The `util.pathname` method produces a pathname out of a directory name `dir` and a filename `file` and returns it.

```
3327 function util.pathname(dir, file)
3328 if #dir == 0 then
3329 return file
3330 else
3331 return dir .. "/" .. file
3332 end
3333 end
```

### 3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
3334 local entities = {}
3335
3336 local character_entities = {
3337 ["Tab"] = 9,
3338 ["NewLine"] = 10,
3339 ["excl"] = 33,
3340 ["QUOT"] = 34,
3341 ["quot"] = 34,
3342 ["num"] = 35,
3343 ["dollar"] = 36,
3344 ["percent"] = 37,
```

3345 ["AMP"] = 38,  
3346 ["amp"] = 38,  
3347 ["apos"] = 39,  
3348 ["lpar"] = 40,  
3349 ["rpar"] = 41,  
3350 ["ast"] = 42,  
3351 ["midast"] = 42,  
3352 ["plus"] = 43,  
3353 ["comma"] = 44,  
3354 ["period"] = 46,  
3355 ["sol"] = 47,  
3356 ["colon"] = 58,  
3357 ["semi"] = 59,  
3358 ["LT"] = 60,  
3359 ["lt"] = 60,  
3360 ["nvlT"] = {60, 8402},  
3361 ["bne"] = {61, 8421},  
3362 ["equals"] = 61,  
3363 ["GT"] = 62,  
3364 ["gt"] = 62,  
3365 ["nvgt"] = {62, 8402},  
3366 ["quest"] = 63,  
3367 ["commat"] = 64,  
3368 ["lbrack"] = 91,  
3369 ["lsqb"] = 91,  
3370 ["bsol"] = 92,  
3371 ["rbrack"] = 93,  
3372 ["rsqb"] = 93,  
3373 ["Hat"] = 94,  
3374 ["UnderBar"] = 95,  
3375 ["lowbar"] = 95,  
3376 ["DiacriticalGrave"] = 96,  
3377 ["grave"] = 96,  
3378 ["fjlig"] = {102, 106},  
3379 ["lbrace"] = 123,  
3380 ["lcub"] = 123,  
3381 ["VerticalLine"] = 124,  
3382 ["verbar"] = 124,  
3383 ["vert"] = 124,  
3384 ["rbrace"] = 125,  
3385 ["rcub"] = 125,  
3386 ["NonBreakingSpace"] = 160,  
3387 ["nbsp"] = 160,  
3388 ["iexcl"] = 161,  
3389 ["cent"] = 162,  
3390 ["pound"] = 163,  
3391 ["curren"] = 164,

3392 ["yen"] = 165,  
3393 ["brvbar"] = 166,  
3394 ["sect"] = 167,  
3395 ["Dot"] = 168,  
3396 ["DoubleDot"] = 168,  
3397 ["die"] = 168,  
3398 ["uml"] = 168,  
3399 ["COPY"] = 169,  
3400 ["copy"] = 169,  
3401 ["ordf"] = 170,  
3402 ["laquo"] = 171,  
3403 ["not"] = 172,  
3404 ["shy"] = 173,  
3405 ["REG"] = 174,  
3406 ["circledR"] = 174,  
3407 ["reg"] = 174,  
3408 ["macr"] = 175,  
3409 ["strns"] = 175,  
3410 ["deg"] = 176,  
3411 ["PlusMinus"] = 177,  
3412 ["plusmn"] = 177,  
3413 ["pm"] = 177,  
3414 ["sup2"] = 178,  
3415 ["sup3"] = 179,  
3416 ["DiacriticalAcute"] = 180,  
3417 ["acute"] = 180,  
3418 ["micro"] = 181,  
3419 ["para"] = 182,  
3420 ["CenterDot"] = 183,  
3421 ["centerdot"] = 183,  
3422 ["middot"] = 183,  
3423 ["Cedilla"] = 184,  
3424 ["cedil"] = 184,  
3425 ["sup1"] = 185,  
3426 ["ordm"] = 186,  
3427 ["raquo"] = 187,  
3428 ["frac14"] = 188,  
3429 ["frac12"] = 189,  
3430 ["half"] = 189,  
3431 ["frac34"] = 190,  
3432 ["iquest"] = 191,  
3433 ["Agrave"] = 192,  
3434 ["Aacute"] = 193,  
3435 ["Acirc"] = 194,  
3436 ["Atilde"] = 195,  
3437 ["Auml"] = 196,  
3438 ["Aring"] = 197,

3439 ["angst"] = 197,  
3440 ["AElig"] = 198,  
3441 ["Ccedil"] = 199,  
3442 ["Egrave"] = 200,  
3443 ["Eacute"] = 201,  
3444 ["Ecirc"] = 202,  
3445 ["Euml"] = 203,  
3446 ["Igrave"] = 204,  
3447 ["Iacute"] = 205,  
3448 ["Icirc"] = 206,  
3449 ["Iuml"] = 207,  
3450 ["ETH"] = 208,  
3451 ["Ntilde"] = 209,  
3452 ["Ograve"] = 210,  
3453 ["Oacute"] = 211,  
3454 ["Ocirc"] = 212,  
3455 ["Otilde"] = 213,  
3456 ["Ouml"] = 214,  
3457 ["times"] = 215,  
3458 ["Oslash"] = 216,  
3459 ["Ugrave"] = 217,  
3460 ["Uacute"] = 218,  
3461 ["Ucirc"] = 219,  
3462 ["Uuml"] = 220,  
3463 ["Yacute"] = 221,  
3464 ["THORN"] = 222,  
3465 ["szlig"] = 223,  
3466 ["agrave"] = 224,  
3467 ["aacute"] = 225,  
3468 ["acirc"] = 226,  
3469 ["atilde"] = 227,  
3470 ["auml"] = 228,  
3471 ["aring"] = 229,  
3472 ["aelig"] = 230,  
3473 ["ccedil"] = 231,  
3474 ["egrave"] = 232,  
3475 ["eacute"] = 233,  
3476 ["ecirc"] = 234,  
3477 ["euml"] = 235,  
3478 ["igrave"] = 236,  
3479 ["iacute"] = 237,  
3480 ["icirc"] = 238,  
3481 ["iuml"] = 239,  
3482 ["eth"] = 240,  
3483 ["ntilde"] = 241,  
3484 ["ograve"] = 242,  
3485 ["oacute"] = 243,

3486 ["ocirc"] = 244,  
3487 ["otilde"] = 245,  
3488 ["ouml"] = 246,  
3489 ["div"] = 247,  
3490 ["divide"] = 247,  
3491 ["oslash"] = 248,  
3492 ["ugrave"] = 249,  
3493 ["uacute"] = 250,  
3494 ["ucirc"] = 251,  
3495 ["uuml"] = 252,  
3496 ["yacute"] = 253,  
3497 ["thorn"] = 254,  
3498 ["yuml"] = 255,  
3499 ["Amacr"] = 256,  
3500 ["amacr"] = 257,  
3501 ["Abreve"] = 258,  
3502 ["abreve"] = 259,  
3503 ["Aogon"] = 260,  
3504 ["aogon"] = 261,  
3505 ["Cacute"] = 262,  
3506 ["cacute"] = 263,  
3507 ["Ccirc"] = 264,  
3508 ["ccirc"] = 265,  
3509 ["Cdot"] = 266,  
3510 ["cdot"] = 267,  
3511 ["Ccaron"] = 268,  
3512 ["ccaron"] = 269,  
3513 ["Dcaron"] = 270,  
3514 ["dcaron"] = 271,  
3515 ["Dstrok"] = 272,  
3516 ["dstrok"] = 273,  
3517 ["Emacr"] = 274,  
3518 ["emacr"] = 275,  
3519 ["Edot"] = 278,  
3520 ["edot"] = 279,  
3521 ["Eogon"] = 280,  
3522 ["eogon"] = 281,  
3523 ["Ecaron"] = 282,  
3524 ["ecaron"] = 283,  
3525 ["Gcirc"] = 284,  
3526 ["gcirc"] = 285,  
3527 ["Gbreve"] = 286,  
3528 ["gbreve"] = 287,  
3529 ["Gdot"] = 288,  
3530 ["gdot"] = 289,  
3531 ["Gcedil"] = 290,  
3532 ["Hcirc"] = 292,

3533 ["hcirc"] = 293,  
3534 ["Hstrokh"] = 294,  
3535 ["hstrokh"] = 295,  
3536 ["Itilde"] = 296,  
3537 ["itilde"] = 297,  
3538 ["Imacr"] = 298,  
3539 ["imacr"] = 299,  
3540 ["Iogon"] = 302,  
3541 ["iogon"] = 303,  
3542 ["Idot"] = 304,  
3543 ["imath"] = 305,  
3544 ["inodot"] = 305,  
3545 ["IJlig"] = 306,  
3546 ["ijlig"] = 307,  
3547 ["Jcirc"] = 308,  
3548 ["jcirc"] = 309,  
3549 ["Kcedil"] = 310,  
3550 ["kcedil"] = 311,  
3551 ["kgreen"] = 312,  
3552 ["Lacute"] = 313,  
3553 ["lacute"] = 314,  
3554 ["Lcedil"] = 315,  
3555 ["lcedil"] = 316,  
3556 ["Lcaron"] = 317,  
3557 ["lcaron"] = 318,  
3558 ["Lmidot"] = 319,  
3559 ["lmidot"] = 320,  
3560 ["Lstrokh"] = 321,  
3561 ["lstrokh"] = 322,  
3562 ["Nacute"] = 323,  
3563 ["nacute"] = 324,  
3564 ["Ncedil"] = 325,  
3565 ["ncedil"] = 326,  
3566 ["Ncaron"] = 327,  
3567 ["ncaron"] = 328,  
3568 ["napos"] = 329,  
3569 ["ENG"] = 330,  
3570 ["eng"] = 331,  
3571 ["Omacr"] = 332,  
3572 ["omacr"] = 333,  
3573 ["Odblac"] = 336,  
3574 ["odblac"] = 337,  
3575 ["OElig"] = 338,  
3576 ["oelig"] = 339,  
3577 ["Racute"] = 340,  
3578 ["racute"] = 341,  
3579 ["Rcedil"] = 342,



3580 ["rcedil"] = 343,  
3581 ["Rcaron"] = 344,  
3582 ["rcaron"] = 345,  
3583 ["Sacute"] = 346,  
3584 ["sacute"] = 347,  
3585 ["Scirc"] = 348,  
3586 ["scirc"] = 349,  
3587 ["Scedil"] = 350,  
3588 ["scedil"] = 351,  
3589 ["Scaron"] = 352,  
3590 ["scaron"] = 353,  
3591 ["Tcedil"] = 354,  
3592 ["tcedil"] = 355,  
3593 ["Tcaron"] = 356,  
3594 ["tcaron"] = 357,  
3595 ["Tstrok"] = 358,  
3596 ["tstrok"] = 359,  
3597 ["Utilde"] = 360,  
3598 ["utilde"] = 361,  
3599 ["Umacr"] = 362,  
3600 ["umacr"] = 363,  
3601 ["Ubreve"] = 364,  
3602 ["ubreve"] = 365,  
3603 ["Uring"] = 366,  
3604 ["uring"] = 367,  
3605 ["Udblac"] = 368,  
3606 ["udblac"] = 369,  
3607 ["Uogon"] = 370,  
3608 ["uogon"] = 371,  
3609 ["Wcirc"] = 372,  
3610 ["wcirc"] = 373,  
3611 ["Ycirc"] = 374,  
3612 ["ycirc"] = 375,  
3613 ["Yuml"] = 376,  
3614 ["Zacute"] = 377,  
3615 ["zacute"] = 378,  
3616 ["Zdot"] = 379,  
3617 ["zdot"] = 380,  
3618 ["Zcaron"] = 381,  
3619 ["zcaron"] = 382,  
3620 ["fnof"] = 402,  
3621 ["imped"] = 437,  
3622 ["gacute"] = 501,  
3623 ["jmath"] = 567,  
3624 ["circ"] = 710,  
3625 ["Hacek"] = 711,  
3626 ["caron"] = 711,

3627 ["Breve"] = 728,  
3628 ["breve"] = 728,  
3629 ["DiacriticalDot"] = 729,  
3630 ["dot"] = 729,  
3631 ["ring"] = 730,  
3632 ["ogon"] = 731,  
3633 ["DiacriticalTilde"] = 732,  
3634 ["tilde"] = 732,  
3635 ["DiacriticalDoubleAcute"] = 733,  
3636 ["dblac"] = 733,  
3637 ["DownBreve"] = 785,  
3638 ["Alpha"] = 913,  
3639 ["Beta"] = 914,  
3640 ["Gamma"] = 915,  
3641 ["Delta"] = 916,  
3642 ["Epsilon"] = 917,  
3643 ["Zeta"] = 918,  
3644 ["Eta"] = 919,  
3645 ["Theta"] = 920,  
3646 ["Iota"] = 921,  
3647 ["Kappa"] = 922,  
3648 ["Lambda"] = 923,  
3649 ["Mu"] = 924,  
3650 ["Nu"] = 925,  
3651 ["Xi"] = 926,  
3652 ["Omicron"] = 927,  
3653 ["Pi"] = 928,  
3654 ["Rho"] = 929,  
3655 ["Sigma"] = 931,  
3656 ["Tau"] = 932,  
3657 ["Upsilon"] = 933,  
3658 ["Phi"] = 934,  
3659 ["Chi"] = 935,  
3660 ["Psi"] = 936,  
3661 ["Omega"] = 937,  
3662 ["ohm"] = 937,  
3663 ["alpha"] = 945,  
3664 ["beta"] = 946,  
3665 ["gamma"] = 947,  
3666 ["delta"] = 948,  
3667 ["epsi"] = 949,  
3668 ["epsilon"] = 949,  
3669 ["zeta"] = 950,  
3670 ["eta"] = 951,  
3671 ["theta"] = 952,  
3672 ["iota"] = 953,  
3673 ["kappa"] = 954,

3674 ["lambda"] = 955,  
3675 ["mu"] = 956,  
3676 ["nu"] = 957,  
3677 ["xi"] = 958,  
3678 ["omicron"] = 959,  
3679 ["pi"] = 960,  
3680 ["rho"] = 961,  
3681 ["sigmaf"] = 962,  
3682 ["sigmav"] = 962,  
3683 ["varsigma"] = 962,  
3684 ["sigma"] = 963,  
3685 ["tau"] = 964,  
3686 ["upsilon"] = 965,  
3687 ["upsilon"] = 965,  
3688 ["phi"] = 966,  
3689 ["chi"] = 967,  
3690 ["psi"] = 968,  
3691 ["omega"] = 969,  
3692 ["thetasym"] = 977,  
3693 ["thetav"] = 977,  
3694 ["vartheta"] = 977,  
3695 ["Upsilon"] = 978,  
3696 ["upsih"] = 978,  
3697 ["phiv"] = 981,  
3698 ["straightphi"] = 981,  
3699 ["varphi"] = 981,  
3700 ["piv"] = 982,  
3701 ["varpi"] = 982,  
3702 ["Gammad"] = 988,  
3703 ["digamma"] = 989,  
3704 ["gammad"] = 989,  
3705 ["kappav"] = 1008,  
3706 ["varkappa"] = 1008,  
3707 ["rhov"] = 1009,  
3708 ["varrho"] = 1009,  
3709 ["epsiv"] = 1013,  
3710 ["straightepsilon"] = 1013,  
3711 ["varepsilon"] = 1013,  
3712 ["backepsilon"] = 1014,  
3713 ["bepsi"] = 1014,  
3714 ["IOcy"] = 1025,  
3715 ["DJcy"] = 1026,  
3716 ["GJcy"] = 1027,  
3717 ["Jukcy"] = 1028,  
3718 ["DScy"] = 1029,  
3719 ["Iukcy"] = 1030,  
3720 ["YIcy"] = 1031,

3721 ["Jsercy"] = 1032,  
3722 ["LJcy"] = 1033,  
3723 ["NJcy"] = 1034,  
3724 ["TSHcy"] = 1035,  
3725 ["KJcy"] = 1036,  
3726 ["Ubrcy"] = 1038,  
3727 ["DZcy"] = 1039,  
3728 ["Acy"] = 1040,  
3729 ["Bcy"] = 1041,  
3730 ["Vcy"] = 1042,  
3731 ["Gcy"] = 1043,  
3732 ["Dcy"] = 1044,  
3733 ["IEcy"] = 1045,  
3734 ["ZHcy"] = 1046,  
3735 ["Zcy"] = 1047,  
3736 ["Icy"] = 1048,  
3737 ["Jcy"] = 1049,  
3738 ["Kcy"] = 1050,  
3739 ["Lcy"] = 1051,  
3740 ["Mcy"] = 1052,  
3741 ["Ncy"] = 1053,  
3742 ["Ocy"] = 1054,  
3743 ["Pcy"] = 1055,  
3744 ["Rcy"] = 1056,  
3745 ["Scy"] = 1057,  
3746 ["Tcy"] = 1058,  
3747 ["Ucy"] = 1059,  
3748 ["Fcy"] = 1060,  
3749 ["KHcy"] = 1061,  
3750 ["TScy"] = 1062,  
3751 ["CHcy"] = 1063,  
3752 ["SHcy"] = 1064,  
3753 ["SHCHcy"] = 1065,  
3754 ["HARDcy"] = 1066,  
3755 ["Ycy"] = 1067,  
3756 ["SOFTcy"] = 1068,  
3757 ["Ecy"] = 1069,  
3758 ["YUcy"] = 1070,  
3759 ["YAcy"] = 1071,  
3760 ["acy"] = 1072,  
3761 ["bcy"] = 1073,  
3762 ["vcy"] = 1074,  
3763 ["gcy"] = 1075,  
3764 ["dcy"] = 1076,  
3765 ["iecy"] = 1077,  
3766 ["zhcy"] = 1078,  
3767 ["zcy"] = 1079,

3768 ["icy"] = 1080,  
3769 ["jcy"] = 1081,  
3770 ["kcy"] = 1082,  
3771 ["lcy"] = 1083,  
3772 ["mcy"] = 1084,  
3773 ["ncy"] = 1085,  
3774 ["ocy"] = 1086,  
3775 ["pcy"] = 1087,  
3776 ["rcy"] = 1088,  
3777 ["scy"] = 1089,  
3778 ["tcy"] = 1090,  
3779 ["ucy"] = 1091,  
3780 ["fcy"] = 1092,  
3781 ["khcy"] = 1093,  
3782 ["tscy"] = 1094,  
3783 ["chcy"] = 1095,  
3784 ["shcy"] = 1096,  
3785 ["shchcy"] = 1097,  
3786 ["hardcy"] = 1098,  
3787 ["ycy"] = 1099,  
3788 ["softcy"] = 1100,  
3789 ["ecy"] = 1101,  
3790 ["yucy"] = 1102,  
3791 ["yacy"] = 1103,  
3792 ["iocy"] = 1105,  
3793 ["djcy"] = 1106,  
3794 ["gjcy"] = 1107,  
3795 ["jukcy"] = 1108,  
3796 ["dscy"] = 1109,  
3797 ["iukcy"] = 1110,  
3798 ["yicy"] = 1111,  
3799 ["jsercy"] = 1112,  
3800 ["ljcy"] = 1113,  
3801 ["njcy"] = 1114,  
3802 ["tshcy"] = 1115,  
3803 ["kjcy"] = 1116,  
3804 ["ubrcy"] = 1118,  
3805 ["dzcy"] = 1119,  
3806 ["ensp"] = 8194,  
3807 ["emsp"] = 8195,  
3808 ["emsp13"] = 8196,  
3809 ["emsp14"] = 8197,  
3810 ["numsp"] = 8199,  
3811 ["puncsp"] = 8200,  
3812 ["ThinSpace"] = 8201,  
3813 ["thinsp"] = 8201,  
3814 ["VeryThinSpace"] = 8202,

3815 ["hairsp"] = 8202,  
3816 ["NegativeMediumSpace"] = 8203,  
3817 ["NegativeThickSpace"] = 8203,  
3818 ["NegativeThinSpace"] = 8203,  
3819 ["NegativeVeryThinSpace"] = 8203,  
3820 ["ZeroWidthSpace"] = 8203,  
3821 ["zwnj"] = 8204,  
3822 ["zwj"] = 8205,  
3823 ["lrm"] = 8206,  
3824 ["rlm"] = 8207,  
3825 ["dash"] = 8208,  
3826 ["hyphen"] = 8208,  
3827 ["ndash"] = 8211,  
3828 ["mdash"] = 8212,  
3829 ["horbar"] = 8213,  
3830 ["Verbar"] = 8214,  
3831 ["Vert"] = 8214,  
3832 ["OpenCurlyQuote"] = 8216,  
3833 ["lsquo"] = 8216,  
3834 ["CloseCurlyQuote"] = 8217,  
3835 ["rsquo"] = 8217,  
3836 ["rsquor"] = 8217,  
3837 ["lsquor"] = 8218,  
3838 ["sbquo"] = 8218,  
3839 ["OpenCurlyDoubleQuote"] = 8220,  
3840 ["ldquo"] = 8220,  
3841 ["CloseCurlyDoubleQuote"] = 8221,  
3842 ["rdquo"] = 8221,  
3843 ["rdquor"] = 8221,  
3844 ["bdquo"] = 8222,  
3845 ["ldquor"] = 8222,  
3846 ["dagger"] = 8224,  
3847 ["Dagger"] = 8225,  
3848 ["ddagger"] = 8225,  
3849 ["bull"] = 8226,  
3850 ["bullet"] = 8226,  
3851 ["nldr"] = 8229,  
3852 ["hellip"] = 8230,  
3853 ["mldr"] = 8230,  
3854 ["permil"] = 8240,  
3855 ["pertenk"] = 8241,  
3856 ["prime"] = 8242,  
3857 ["Prime"] = 8243,  
3858 ["tprime"] = 8244,  
3859 ["backprime"] = 8245,  
3860 ["bprime"] = 8245,  
3861 ["lsaquo"] = 8249,

3862 ["rsaquo"] = 8250,  
3863 ["OverBar"] = 8254,  
3864 ["oline"] = 8254,  
3865 ["caret"] = 8257,  
3866 ["hybull"] = 8259,  
3867 ["frasl"] = 8260,  
3868 ["bsemi"] = 8271,  
3869 ["qprime"] = 8279,  
3870 ["MediumSpace"] = 8287,  
3871 ["ThickSpace"] = {8287, 8202},  
3872 ["NoBreak"] = 8288,  
3873 ["ApplyFunction"] = 8289,  
3874 ["af"] = 8289,  
3875 ["InvisibleTimes"] = 8290,  
3876 ["it"] = 8290,  
3877 ["InvisibleComma"] = 8291,  
3878 ["ic"] = 8291,  
3879 ["euro"] = 8364,  
3880 ["TripleDot"] = 8411,  
3881 ["tdot"] = 8411,  
3882 ["DotDot"] = 8412,  
3883 ["Copf"] = 8450,  
3884 ["complexes"] = 8450,  
3885 ["incare"] = 8453,  
3886 ["gscr"] = 8458,  
3887 ["HilbertSpace"] = 8459,  
3888 ["Hscr"] = 8459,  
3889 ["hamilt"] = 8459,  
3890 ["Hfr"] = 8460,  
3891 ["Poincareplane"] = 8460,  
3892 ["Hopf"] = 8461,  
3893 ["quaternions"] = 8461,  
3894 ["planckh"] = 8462,  
3895 ["hbar"] = 8463,  
3896 ["hslash"] = 8463,  
3897 ["planck"] = 8463,  
3898 ["plankv"] = 8463,  
3899 ["Iscr"] = 8464,  
3900 ["imagline"] = 8464,  
3901 ["Ifr"] = 8465,  
3902 ["Im"] = 8465,  
3903 ["image"] = 8465,  
3904 ["imagpart"] = 8465,  
3905 ["Laplacetrif"] = 8466,  
3906 ["Lscr"] = 8466,  
3907 ["lagran"] = 8466,  
3908 ["ell"] = 8467,

3909 ["Nopf"] = 8469,  
3910 ["naturals"] = 8469,  
3911 ["numero"] = 8470,  
3912 ["copysr"] = 8471,  
3913 ["weierp"] = 8472,  
3914 ["wp"] = 8472,  
3915 ["Popf"] = 8473,  
3916 ["primes"] = 8473,  
3917 ["Qopf"] = 8474,  
3918 ["rationals"] = 8474,  
3919 ["Rscr"] = 8475,  
3920 ["realine"] = 8475,  
3921 ["Re"] = 8476,  
3922 ["Rfr"] = 8476,  
3923 ["real"] = 8476,  
3924 ["realpart"] = 8476,  
3925 ["Ropf"] = 8477,  
3926 ["reals"] = 8477,  
3927 ["rx"] = 8478,  
3928 ["TRADE"] = 8482,  
3929 ["trade"] = 8482,  
3930 ["Zopf"] = 8484,  
3931 ["integers"] = 8484,  
3932 ["mho"] = 8487,  
3933 ["Zfr"] = 8488,  
3934 ["zeetrf"] = 8488,  
3935 ["iiota"] = 8489,  
3936 ["Bernoullis"] = 8492,  
3937 ["Bscr"] = 8492,  
3938 ["bernou"] = 8492,  
3939 ["Cayleys"] = 8493,  
3940 ["Cfr"] = 8493,  
3941 ["escr"] = 8495,  
3942 ["Escr"] = 8496,  
3943 ["expectation"] = 8496,  
3944 ["Fouriertrf"] = 8497,  
3945 ["Fscr"] = 8497,  
3946 ["Mellintrf"] = 8499,  
3947 ["Mscr"] = 8499,  
3948 ["phmmat"] = 8499,  
3949 ["order"] = 8500,  
3950 ["orderof"] = 8500,  
3951 ["oscr"] = 8500,  
3952 ["alefsym"] = 8501,  
3953 ["aleph"] = 8501,  
3954 ["beth"] = 8502,  
3955 ["gimel"] = 8503,



3956 ["daleth"] = 8504,  
 3957 ["CapitalDifferentialD"] = 8517,  
 3958 ["DD"] = 8517,  
 3959 ["DifferentialD"] = 8518,  
 3960 ["dd"] = 8518,  
 3961 ["ExponentialE"] = 8519,  
 3962 ["ee"] = 8519,  
 3963 ["exponentiale"] = 8519,  
 3964 ["ImaginaryI"] = 8520,  
 3965 ["ii"] = 8520,  
 3966 ["frac13"] = 8531,  
 3967 ["frac23"] = 8532,  
 3968 ["frac15"] = 8533,  
 3969 ["frac25"] = 8534,  
 3970 ["frac35"] = 8535,  
 3971 ["frac45"] = 8536,  
 3972 ["frac16"] = 8537,  
 3973 ["frac56"] = 8538,  
 3974 ["frac18"] = 8539,  
 3975 ["frac38"] = 8540,  
 3976 ["frac58"] = 8541,  
 3977 ["frac78"] = 8542,  
 3978 ["LeftArrow"] = 8592,  
 3979 ["ShortLeftArrow"] = 8592,  
 3980 ["larr"] = 8592,  
 3981 ["leftarrow"] = 8592,  
 3982 ["slarr"] = 8592,  
 3983 ["ShortUpArrow"] = 8593,  
 3984 ["UpArrow"] = 8593,  
 3985 ["uarr"] = 8593,  
 3986 ["uparrow"] = 8593,  
 3987 ["RightArrow"] = 8594,  
 3988 ["ShortRightArrow"] = 8594,  
 3989 ["rarr"] = 8594,  
 3990 ["rightarrow"] = 8594,  
 3991 ["srarr"] = 8594,  
 3992 ["DownArrow"] = 8595,  
 3993 ["ShortDownArrow"] = 8595,  
 3994 ["darr"] = 8595,  
 3995 ["downarrow"] = 8595,  
 3996 ["LeftRightArrow"] = 8596,  
 3997 ["harr"] = 8596,  
 3998 ["leftrightarrow"] = 8596,  
 3999 ["UpDownArrow"] = 8597,  
 4000 ["updownarrow"] = 8597,  
 4001 ["varr"] = 8597,  
 4002 ["UpperLeftArrow"] = 8598,

4003 ["nwarr"] = 8598,  
 4004 ["nwarrow"] = 8598,  
 4005 ["UpperRightArrow"] = 8599,  
 4006 ["nearr"] = 8599,  
 4007 ["nearrow"] = 8599,  
 4008 ["LowerRightArrow"] = 8600,  
 4009 ["searr"] = 8600,  
 4010 ["searrow"] = 8600,  
 4011 ["LowerLeftArrow"] = 8601,  
 4012 ["swarr"] = 8601,  
 4013 ["swarrow"] = 8601,  
 4014 ["nlarr"] = 8602,  
 4015 ["nleftarrow"] = 8602,  
 4016 ["nrarr"] = 8603,  
 4017 ["nrightarrow"] = 8603,  
 4018 ["nrarrw"] = {8605, 824},  
 4019 ["rarrw"] = 8605,  
 4020 ["rightsquigarrow"] = 8605,  
 4021 ["Larr"] = 8606,  
 4022 ["twoheadleftarrow"] = 8606,  
 4023 ["Uarr"] = 8607,  
 4024 ["Rarr"] = 8608,  
 4025 ["twoheadrightarrow"] = 8608,  
 4026 ["Darr"] = 8609,  
 4027 ["larrtl"] = 8610,  
 4028 ["leftarrowtail"] = 8610,  
 4029 ["rarrtl"] = 8611,  
 4030 ["rightarrowtail"] = 8611,  
 4031 ["LeftTeeArrow"] = 8612,  
 4032 ["mapstoleft"] = 8612,  
 4033 ["UpTeeArrow"] = 8613,  
 4034 ["mapstoup"] = 8613,  
 4035 ["RightTeeArrow"] = 8614,  
 4036 ["map"] = 8614,  
 4037 ["mapsto"] = 8614,  
 4038 ["DownTeeArrow"] = 8615,  
 4039 ["mapstodown"] = 8615,  
 4040 ["hookleftarrow"] = 8617,  
 4041 ["larrhk"] = 8617,  
 4042 ["hookrightarrow"] = 8618,  
 4043 ["rarrhk"] = 8618,  
 4044 ["larrlp"] = 8619,  
 4045 ["looparrowleft"] = 8619,  
 4046 ["looparrowright"] = 8620,  
 4047 ["rarrlp"] = 8620,  
 4048 ["harrw"] = 8621,  
 4049 ["leftrightsquigarrow"] = 8621,

4050 ["nharr"] = 8622,  
 4051 ["nleftrightharrow"] = 8622,  
 4052 ["Lsh"] = 8624,  
 4053 ["lsh"] = 8624,  
 4054 ["Rsh"] = 8625,  
 4055 ["rsh"] = 8625,  
 4056 ["ldsh"] = 8626,  
 4057 ["rdsh"] = 8627,  
 4058 ["crarr"] = 8629,  
 4059 ["cularr"] = 8630,  
 4060 ["curvearrowleft"] = 8630,  
 4061 ["curarr"] = 8631,  
 4062 ["curvearrowright"] = 8631,  
 4063 ["circlearrowleft"] = 8634,  
 4064 ["olarr"] = 8634,  
 4065 ["circlearrowright"] = 8635,  
 4066 ["orarr"] = 8635,  
 4067 ["LeftVector"] = 8636,  
 4068 ["leftharpoonup"] = 8636,  
 4069 ["lharu"] = 8636,  
 4070 ["DownLeftVector"] = 8637,  
 4071 ["leftharpoondown"] = 8637,  
 4072 ["lhard"] = 8637,  
 4073 ["RightUpVector"] = 8638,  
 4074 ["uharr"] = 8638,  
 4075 ["upharpoonright"] = 8638,  
 4076 ["LeftUpVector"] = 8639,  
 4077 ["uharl"] = 8639,  
 4078 ["upharpoonleft"] = 8639,  
 4079 ["RightVector"] = 8640,  
 4080 ["rharu"] = 8640,  
 4081 ["rightharpoonup"] = 8640,  
 4082 ["DownRightVector"] = 8641,  
 4083 ["rhard"] = 8641,  
 4084 ["rightharpoondown"] = 8641,  
 4085 ["RightDownVector"] = 8642,  
 4086 ["dharr"] = 8642,  
 4087 ["downharpoonright"] = 8642,  
 4088 ["LeftDownVector"] = 8643,  
 4089 ["dharl"] = 8643,  
 4090 ["downharpoonleft"] = 8643,  
 4091 ["RightArrowLeftArrow"] = 8644,  
 4092 ["rightleftarrows"] = 8644,  
 4093 ["rlarr"] = 8644,  
 4094 ["UpArrowDownArrow"] = 8645,  
 4095 ["udarr"] = 8645,  
 4096 ["LeftArrowRightArrow"] = 8646,

4097 ["leftrightharpoons"] = 8646,  
 4098 ["larr"] = 8646,  
 4099 ["leftleftarrows"] = 8647,  
 4100 ["llarr"] = 8647,  
 4101 ["uparrows"] = 8648,  
 4102 ["uarr"] = 8648,  
 4103 ["rightrightarrows"] = 8649,  
 4104 ["rrarr"] = 8649,  
 4105 ["ddarr"] = 8650,  
 4106 ["downdownarrows"] = 8650,  
 4107 ["ReverseEquilibrium"] = 8651,  
 4108 ["leftrightharpoons"] = 8651,  
 4109 ["lrhar"] = 8651,  
 4110 ["Equilibrium"] = 8652,  
 4111 ["rightleftharpoons"] = 8652,  
 4112 ["rlhar"] = 8652,  
 4113 ["nLeftarrow"] = 8653,  
 4114 ["nlArr"] = 8653,  
 4115 ["nLeftrightarrow"] = 8654,  
 4116 ["nhArr"] = 8654,  
 4117 ["nRightarrow"] = 8655,  
 4118 ["nrArr"] = 8655,  
 4119 ["DoubleLeftArrow"] = 8656,  
 4120 ["Leftarrow"] = 8656,  
 4121 ["lArr"] = 8656,  
 4122 ["DoubleUpArrow"] = 8657,  
 4123 ["Uparrow"] = 8657,  
 4124 ["uArr"] = 8657,  
 4125 ["DoubleRightArrow"] = 8658,  
 4126 ["Implies"] = 8658,  
 4127 ["Rightarrow"] = 8658,  
 4128 ["rArr"] = 8658,  
 4129 ["DoubleDownArrow"] = 8659,  
 4130 ["Downarrow"] = 8659,  
 4131 ["dArr"] = 8659,  
 4132 ["DoubleLeftRightArrow"] = 8660,  
 4133 ["Leftrightarrow"] = 8660,  
 4134 ["hArr"] = 8660,  
 4135 ["iff"] = 8660,  
 4136 ["DoubleUpDownArrow"] = 8661,  
 4137 ["Updownarrow"] = 8661,  
 4138 ["vArr"] = 8661,  
 4139 ["nwArr"] = 8662,  
 4140 ["neArr"] = 8663,  
 4141 ["seArr"] = 8664,  
 4142 ["swArr"] = 8665,  
 4143 ["Lleftarrow"] = 8666,

4144 ["lAarr"] = 8666,  
4145 ["Rightarrow"] = 8667,  
4146 ["rAarr"] = 8667,  
4147 ["zigrarr"] = 8669,  
4148 ["LeftArrowBar"] = 8676,  
4149 ["larrb"] = 8676,  
4150 ["RightArrowBar"] = 8677,  
4151 ["rarrb"] = 8677,  
4152 ["DownArrowUpArrow"] = 8693,  
4153 ["duarr"] = 8693,  
4154 ["loarr"] = 8701,  
4155 ["roarr"] = 8702,  
4156 ["hoarr"] = 8703,  
4157 ["ForAll"] = 8704,  
4158 ["forall"] = 8704,  
4159 ["comp"] = 8705,  
4160 ["complement"] = 8705,  
4161 ["PartialD"] = 8706,  
4162 ["npart"] = {8706, 824},  
4163 ["part"] = 8706,  
4164 ["Exists"] = 8707,  
4165 ["exist"] = 8707,  
4166 ["NotExists"] = 8708,  
4167 ["nexist"] = 8708,  
4168 ["nexists"] = 8708,  
4169 ["empty"] = 8709,  
4170 ["emptyset"] = 8709,  
4171 ["emptyv"] = 8709,  
4172 ["varnothing"] = 8709,  
4173 ["Del"] = 8711,  
4174 ["nabla"] = 8711,  
4175 ["Element"] = 8712,  
4176 ["in"] = 8712,  
4177 ["isin"] = 8712,  
4178 ["isinv"] = 8712,  
4179 ["NotElement"] = 8713,  
4180 ["notin"] = 8713,  
4181 ["notinva"] = 8713,  
4182 ["ReverseElement"] = 8715,  
4183 ["SuchThat"] = 8715,  
4184 ["ni"] = 8715,  
4185 ["niv"] = 8715,  
4186 ["NotReverseElement"] = 8716,  
4187 ["notni"] = 8716,  
4188 ["notniva"] = 8716,  
4189 ["Product"] = 8719,  
4190 ["prod"] = 8719,

4191 ["Coproduct"] = 8720,  
 4192 ["coprod"] = 8720,  
 4193 ["Sum"] = 8721,  
 4194 ["sum"] = 8721,  
 4195 ["minus"] = 8722,  
 4196 ["MinusPlus"] = 8723,  
 4197 ["mnplus"] = 8723,  
 4198 ["mp"] = 8723,  
 4199 ["dotplus"] = 8724,  
 4200 ["plusdo"] = 8724,  
 4201 ["Backslash"] = 8726,  
 4202 ["setminus"] = 8726,  
 4203 ["setmn"] = 8726,  
 4204 ["smallsetminus"] = 8726,  
 4205 ["ssetmn"] = 8726,  
 4206 ["lowast"] = 8727,  
 4207 ["SmallCircle"] = 8728,  
 4208 ["compfn"] = 8728,  
 4209 ["Sqrt"] = 8730,  
 4210 ["radic"] = 8730,  
 4211 ["Proportional"] = 8733,  
 4212 ["prop"] = 8733,  
 4213 ["propto"] = 8733,  
 4214 ["varpropto"] = 8733,  
 4215 ["vprop"] = 8733,  
 4216 ["infin"] = 8734,  
 4217 ["angrt"] = 8735,  
 4218 ["ang"] = 8736,  
 4219 ["angle"] = 8736,  
 4220 ["nang"] = {8736, 8402},  
 4221 ["angmsd"] = 8737,  
 4222 ["measuredangle"] = 8737,  
 4223 ["angsph"] = 8738,  
 4224 ["VerticalBar"] = 8739,  
 4225 ["mid"] = 8739,  
 4226 ["shortmid"] = 8739,  
 4227 ["smid"] = 8739,  
 4228 ["NotVerticalBar"] = 8740,  
 4229 ["nmid"] = 8740,  
 4230 ["nshortmid"] = 8740,  
 4231 ["nsmid"] = 8740,  
 4232 ["DoubleVerticalBar"] = 8741,  
 4233 ["par"] = 8741,  
 4234 ["parallel"] = 8741,  
 4235 ["shortparallel"] = 8741,  
 4236 ["spar"] = 8741,  
 4237 ["NotDoubleVerticalBar"] = 8742,

4238 ["npar"] = 8742,  
 4239 ["nparallel"] = 8742,  
 4240 ["nshortparallel"] = 8742,  
 4241 ["nspar"] = 8742,  
 4242 ["and"] = 8743,  
 4243 ["wedge"] = 8743,  
 4244 ["or"] = 8744,  
 4245 ["vee"] = 8744,  
 4246 ["cap"] = 8745,  
 4247 ["caps"] = {8745, 65024},  
 4248 ["cup"] = 8746,  
 4249 ["cups"] = {8746, 65024},  
 4250 ["Integral"] = 8747,  
 4251 ["int"] = 8747,  
 4252 ["Int"] = 8748,  
 4253 ["iiint"] = 8749,  
 4254 ["tint"] = 8749,  
 4255 ["ContourIntegral"] = 8750,  
 4256 ["conint"] = 8750,  
 4257 ["oint"] = 8750,  
 4258 ["Conint"] = 8751,  
 4259 ["DoubleContourIntegral"] = 8751,  
 4260 ["Cconint"] = 8752,  
 4261 ["cwint"] = 8753,  
 4262 ["ClockwiseContourIntegral"] = 8754,  
 4263 ["cwconint"] = 8754,  
 4264 ["CounterClockwiseContourIntegral"] = 8755,  
 4265 ["awconint"] = 8755,  
 4266 ["Therefore"] = 8756,  
 4267 ["there4"] = 8756,  
 4268 ["therefore"] = 8756,  
 4269 ["Because"] = 8757,  
 4270 ["because"] = 8757,  
 4271 ["because"] = 8757,  
 4272 ["ratio"] = 8758,  
 4273 ["Colon"] = 8759,  
 4274 ["Proportion"] = 8759,  
 4275 ["dotminus"] = 8760,  
 4276 ["minusd"] = 8760,  
 4277 ["mDDot"] = 8762,  
 4278 ["homtht"] = 8763,  
 4279 ["Tilde"] = 8764,  
 4280 ["nvsim"] = {8764, 8402},  
 4281 ["sim"] = 8764,  
 4282 ["thicksim"] = 8764,  
 4283 ["thksim"] = 8764,  
 4284 ["backsim"] = 8765,

4285 ["bsim"] = 8765,  
 4286 ["race"] = {8765, 817},  
 4287 ["ac"] = 8766,  
 4288 ["acE"] = {8766, 819},  
 4289 ["mstpos"] = 8766,  
 4290 ["acd"] = 8767,  
 4291 ["VerticalTilde"] = 8768,  
 4292 ["wr"] = 8768,  
 4293 ["wreath"] = 8768,  
 4294 ["NotTilde"] = 8769,  
 4295 ["nsim"] = 8769,  
 4296 ["EqualTilde"] = 8770,  
 4297 ["NotEqualTilde"] = {8770, 824},  
 4298 ["eqsim"] = 8770,  
 4299 ["esim"] = 8770,  
 4300 ["nesim"] = {8770, 824},  
 4301 ["TildeEqual"] = 8771,  
 4302 ["sime"] = 8771,  
 4303 ["simeq"] = 8771,  
 4304 ["NotTildeEqual"] = 8772,  
 4305 ["nsime"] = 8772,  
 4306 ["nsimeq"] = 8772,  
 4307 ["TildeFullEqual"] = 8773,  
 4308 ["cong"] = 8773,  
 4309 ["simne"] = 8774,  
 4310 ["NotTildeFullEqual"] = 8775,  
 4311 ["ncong"] = 8775,  
 4312 ["TildeTilde"] = 8776,  
 4313 ["ap"] = 8776,  
 4314 ["approx"] = 8776,  
 4315 ["asymp"] = 8776,  
 4316 ["thickapprox"] = 8776,  
 4317 ["thkap"] = 8776,  
 4318 ["NotTildeTilde"] = 8777,  
 4319 ["nap"] = 8777,  
 4320 ["napprox"] = 8777,  
 4321 ["ape"] = 8778,  
 4322 ["approxpeq"] = 8778,  
 4323 ["apid"] = 8779,  
 4324 ["napid"] = {8779, 824},  
 4325 ["backcong"] = 8780,  
 4326 ["bcong"] = 8780,  
 4327 ["CupCap"] = 8781,  
 4328 ["asympeq"] = 8781,  
 4329 ["nvap"] = {8781, 8402},  
 4330 ["Bumpeq"] = 8782,  
 4331 ["HumpDownHump"] = 8782,



4332 ["NotHumpDownHump"] = {8782, 824},  
4333 ["bump"] = 8782,  
4334 ["nbump"] = {8782, 824},  
4335 ["HumpEqual"] = 8783,  
4336 ["NotHumpEqual"] = {8783, 824},  
4337 ["bumpe"] = 8783,  
4338 ["bumpeq"] = 8783,  
4339 ["nbumpe"] = {8783, 824},  
4340 ["DotEqual"] = 8784,  
4341 ["doteq"] = 8784,  
4342 ["esdot"] = 8784,  
4343 ["nedot"] = {8784, 824},  
4344 ["doteqdot"] = 8785,  
4345 ["eDot"] = 8785,  
4346 ["efDot"] = 8786,  
4347 ["fallingdotseq"] = 8786,  
4348 ["erDot"] = 8787,  
4349 ["risingdotseq"] = 8787,  
4350 ["Assign"] = 8788,  
4351 ["colone"] = 8788,  
4352 ["coloneq"] = 8788,  
4353 ["ecolon"] = 8789,  
4354 ["eqcolon"] = 8789,  
4355 ["ecir"] = 8790,  
4356 ["eqcirc"] = 8790,  
4357 ["circeq"] = 8791,  
4358 ["cire"] = 8791,  
4359 ["wedgeq"] = 8793,  
4360 ["veeeq"] = 8794,  
4361 ["triangleq"] = 8796,  
4362 ["trie"] = 8796,  
4363 ["equest"] = 8799,  
4364 ["questeq"] = 8799,  
4365 ["NotEqual"] = 8800,  
4366 ["ne"] = 8800,  
4367 ["Congruent"] = 8801,  
4368 ["bnequiv"] = {8801, 8421},  
4369 ["equiv"] = 8801,  
4370 ["NotCongruent"] = 8802,  
4371 ["nequiv"] = 8802,  
4372 ["le"] = 8804,  
4373 ["leq"] = 8804,  
4374 ["nvle"] = {8804, 8402},  
4375 ["GreaterEqual"] = 8805,  
4376 ["ge"] = 8805,  
4377 ["geq"] = 8805,  
4378 ["nvge"] = {8805, 8402},

4379 ["LessFullEqual"] = 8806,  
4380 ["lE"] = 8806,  
4381 ["leqq"] = 8806,  
4382 ["nlE"] = {8806, 824},  
4383 ["nleqq"] = {8806, 824},  
4384 ["GreaterFullEqual"] = 8807,  
4385 ["NotGreaterFullEqual"] = {8807, 824},  
4386 ["gE"] = 8807,  
4387 ["geqq"] = 8807,  
4388 ["ngE"] = {8807, 824},  
4389 ["ngeqq"] = {8807, 824},  
4390 ["lnE"] = 8808,  
4391 ["lneqq"] = 8808,  
4392 ["lvertneqq"] = {8808, 65024},  
4393 ["lvnE"] = {8808, 65024},  
4394 ["gnE"] = 8809,  
4395 ["gneqq"] = 8809,  
4396 ["gvertneqq"] = {8809, 65024},  
4397 ["gvnE"] = {8809, 65024},  
4398 ["Lt"] = 8810,  
4399 ["NestedLessLess"] = 8810,  
4400 ["NotLessLess"] = {8810, 824},  
4401 ["l1"] = 8810,  
4402 ["nLt"] = {8810, 8402},  
4403 ["nLtv"] = {8810, 824},  
4404 ["Gt"] = 8811,  
4405 ["NestedGreaterGreater"] = 8811,  
4406 ["NotGreaterGreater"] = {8811, 824},  
4407 ["gg"] = 8811,  
4408 ["nGt"] = {8811, 8402},  
4409 ["nGtv"] = {8811, 824},  
4410 ["between"] = 8812,  
4411 ["twixt"] = 8812,  
4412 ["NotCupCap"] = 8813,  
4413 ["NotLess"] = 8814,  
4414 ["nless"] = 8814,  
4415 ["nlt"] = 8814,  
4416 ["NotGreater"] = 8815,  
4417 ["ngt"] = 8815,  
4418 ["ngtr"] = 8815,  
4419 ["NotLessEqual"] = 8816,  
4420 ["nle"] = 8816,  
4421 ["nleq"] = 8816,  
4422 ["NotGreaterEqual"] = 8817,  
4423 ["nge"] = 8817,  
4424 ["ngeq"] = 8817,  
4425 ["LessTilde"] = 8818,

4426 ["lesssim"] = 8818,  
4427 ["lsim"] = 8818,  
4428 ["GreaterTilde"] = 8819,  
4429 ["gsim"] = 8819,  
4430 ["gtrsim"] = 8819,  
4431 ["NotLessTilde"] = 8820,  
4432 ["nlsim"] = 8820,  
4433 ["NotGreaterTilde"] = 8821,  
4434 ["ngsim"] = 8821,  
4435 ["LessGreater"] = 8822,  
4436 ["lessgtr"] = 8822,  
4437 ["lg"] = 8822,  
4438 ["GreaterLess"] = 8823,  
4439 ["gl"] = 8823,  
4440 ["gtrless"] = 8823,  
4441 ["NotLessGreater"] = 8824,  
4442 ["ntlg"] = 8824,  
4443 ["NotGreaterLess"] = 8825,  
4444 ["ntgl"] = 8825,  
4445 ["Precedes"] = 8826,  
4446 ["pr"] = 8826,  
4447 ["prec"] = 8826,  
4448 ["Succeeds"] = 8827,  
4449 ["sc"] = 8827,  
4450 ["succ"] = 8827,  
4451 ["PrecedesSlantEqual"] = 8828,  
4452 ["prcue"] = 8828,  
4453 ["preccurlyeq"] = 8828,  
4454 ["SucceedsSlantEqual"] = 8829,  
4455 ["sccue"] = 8829,  
4456 ["succcurlyeq"] = 8829,  
4457 ["PrecedesTilde"] = 8830,  
4458 ["precsim"] = 8830,  
4459 ["prsim"] = 8830,  
4460 ["NotSucceedsTilde"] = {8831, 824},  
4461 ["SucceedsTilde"] = 8831,  
4462 ["scsim"] = 8831,  
4463 ["succsim"] = 8831,  
4464 ["NotPrecedes"] = 8832,  
4465 ["npr"] = 8832,  
4466 ["nprec"] = 8832,  
4467 ["NotSucceeds"] = 8833,  
4468 ["nsc"] = 8833,  
4469 ["nsucc"] = 8833,  
4470 ["NotSubset"] = {8834, 8402},  
4471 ["nsubset"] = {8834, 8402},  
4472 ["sub"] = 8834,

```

4473 ["subset"] = 8834,
4474 ["vnsup"] = {8834, 8402},
4475 ["NotSuperset"] = {8835, 8402},
4476 ["Superset"] = 8835,
4477 ["nsupset"] = {8835, 8402},
4478 ["sup"] = 8835,
4479 ["supset"] = 8835,
4480 ["vnsup"] = {8835, 8402},
4481 ["nsub"] = 8836,
4482 ["nsup"] = 8837,
4483 ["SubsetEqual"] = 8838,
4484 ["sube"] = 8838,
4485 ["subseteq"] = 8838,
4486 ["SupersetEqual"] = 8839,
4487 ["supe"] = 8839,
4488 ["supseteq"] = 8839,
4489 ["NotSubsetEqual"] = 8840,
4490 ["nsube"] = 8840,
4491 ["nsubseteq"] = 8840,
4492 ["NotSupersetEqual"] = 8841,
4493 ["nsupe"] = 8841,
4494 ["nsupseteq"] = 8841,
4495 ["subne"] = 8842,
4496 ["subsetneq"] = 8842,
4497 ["varsubsetneq"] = {8842, 65024},
4498 ["vsubne"] = {8842, 65024},
4499 ["supne"] = 8843,
4500 ["supsetneq"] = 8843,
4501 ["varsupsetneq"] = {8843, 65024},
4502 ["vsupne"] = {8843, 65024},
4503 ["cupdot"] = 8845,
4504 ["UnionPlus"] = 8846,
4505 ["uplus"] = 8846,
4506 ["NotSquareSubset"] = {8847, 824},
4507 ["SquareSubset"] = 8847,
4508 ["sqsub"] = 8847,
4509 ["sqsubset"] = 8847,
4510 ["NotSquareSuperset"] = {8848, 824},
4511 ["SquareSuperset"] = 8848,
4512 ["sqsup"] = 8848,
4513 ["sqsupset"] = 8848,
4514 ["SquareSubsetEqual"] = 8849,
4515 ["sqsube"] = 8849,
4516 ["sqsubseteq"] = 8849,
4517 ["SquareSupersetEqual"] = 8850,
4518 ["sqsupe"] = 8850,
4519 ["sqsupseteq"] = 8850,

```

```

4520 ["SquareIntersection"] = 8851,
4521 ["sqcap"] = 8851,
4522 ["sqcaps"] = {8851, 65024},
4523 ["SquareUnion"] = 8852,
4524 ["sqcup"] = 8852,
4525 ["sqcups"] = {8852, 65024},
4526 ["CirclePlus"] = 8853,
4527 ["oplus"] = 8853,
4528 ["CircleMinus"] = 8854,
4529 ["ominus"] = 8854,
4530 ["CircleTimes"] = 8855,
4531 ["otimes"] = 8855,
4532 ["osol"] = 8856,
4533 ["CircleDot"] = 8857,
4534 ["odot"] = 8857,
4535 ["circledcirc"] = 8858,
4536 ["ocir"] = 8858,
4537 ["circledast"] = 8859,
4538 ["oast"] = 8859,
4539 ["circleddash"] = 8861,
4540 ["odash"] = 8861,
4541 ["boxplus"] = 8862,
4542 ["plusb"] = 8862,
4543 ["boxminus"] = 8863,
4544 ["minusb"] = 8863,
4545 ["boxtimes"] = 8864,
4546 ["timesb"] = 8864,
4547 ["dotscircle"] = 8865,
4548 ["sdotb"] = 8865,
4549 ["RightTee"] = 8866,
4550 ["vdash"] = 8866,
4551 ["LeftTee"] = 8867,
4552 ["dashv"] = 8867,
4553 ["DownTee"] = 8868,
4554 ["top"] = 8868,
4555 ["UpTee"] = 8869,
4556 ["bot"] = 8869,
4557 ["bottom"] = 8869,
4558 ["perp"] = 8869,
4559 ["models"] = 8871,
4560 ["DoubleRightTee"] = 8872,
4561 ["vDash"] = 8872,
4562 ["Vdash"] = 8873,
4563 ["Vvdash"] = 8874,
4564 ["VDash"] = 8875,
4565 ["nvdash"] = 8876,
4566 ["nvDash"] = 8877,

```

4567 ["nVdash"] = 8878,  
 4568 ["nVDash"] = 8879,  
 4569 ["prurel"] = 8880,  
 4570 ["LeftTriangle"] = 8882,  
 4571 ["vartriangleleft"] = 8882,  
 4572 ["vltri"] = 8882,  
 4573 ["RightTriangle"] = 8883,  
 4574 ["vartriangleright"] = 8883,  
 4575 ["vrtri"] = 8883,  
 4576 ["LeftTriangleEqual"] = 8884,  
 4577 ["ltrie"] = 8884,  
 4578 ["nvltrie"] = {8884, 8402},  
 4579 ["trianglelefteq"] = 8884,  
 4580 ["RightTriangleEqual"] = 8885,  
 4581 ["nvrtrie"] = {8885, 8402},  
 4582 ["rtrie"] = 8885,  
 4583 ["trianglerighteq"] = 8885,  
 4584 ["origof"] = 8886,  
 4585 ["imof"] = 8887,  
 4586 ["multimap"] = 8888,  
 4587 ["mumap"] = 8888,  
 4588 ["hercon"] = 8889,  
 4589 ["intcal"] = 8890,  
 4590 ["intercal"] = 8890,  
 4591 ["veebar"] = 8891,  
 4592 ["barvee"] = 8893,  
 4593 ["angrtvb"] = 8894,  
 4594 ["ltri"] = 8895,  
 4595 ["Wedge"] = 8896,  
 4596 ["bigwedge"] = 8896,  
 4597 ["xwedge"] = 8896,  
 4598 ["Vee"] = 8897,  
 4599 ["bigvee"] = 8897,  
 4600 ["xvee"] = 8897,  
 4601 ["Intersection"] = 8898,  
 4602 ["bigcap"] = 8898,  
 4603 ["xcap"] = 8898,  
 4604 ["Union"] = 8899,  
 4605 ["bigcup"] = 8899,  
 4606 ["xcup"] = 8899,  
 4607 ["Diamond"] = 8900,  
 4608 ["diam"] = 8900,  
 4609 ["diamond"] = 8900,  
 4610 ["sdot"] = 8901,  
 4611 ["Star"] = 8902,  
 4612 ["sstarf"] = 8902,  
 4613 ["divideontimes"] = 8903,

4614 ["divonx"] = 8903,  
4615 ["bowtie"] = 8904,  
4616 ["ltimes"] = 8905,  
4617 ["rtimes"] = 8906,  
4618 ["leftthreetimes"] = 8907,  
4619 ["lthree"] = 8907,  
4620 ["rightthreetimes"] = 8908,  
4621 ["rthree"] = 8908,  
4622 ["backsimeq"] = 8909,  
4623 ["bsime"] = 8909,  
4624 ["curlyvee"] = 8910,  
4625 ["cuvee"] = 8910,  
4626 ["curlywedge"] = 8911,  
4627 ["cuwed"] = 8911,  
4628 ["Sub"] = 8912,  
4629 ["Subset"] = 8912,  
4630 ["Sup"] = 8913,  
4631 ["Supset"] = 8913,  
4632 ["Cap"] = 8914,  
4633 ["Cup"] = 8915,  
4634 ["fork"] = 8916,  
4635 ["pitchfork"] = 8916,  
4636 ["epar"] = 8917,  
4637 ["lessdot"] = 8918,  
4638 ["ltdot"] = 8918,  
4639 ["gtdot"] = 8919,  
4640 ["gtrdot"] = 8919,  
4641 ["Ll"] = 8920,  
4642 ["nLl"] = {8920, 824},  
4643 ["Gg"] = 8921,  
4644 ["ggg"] = 8921,  
4645 ["nGg"] = {8921, 824},  
4646 ["LessEqualGreater"] = 8922,  
4647 ["leg"] = 8922,  
4648 ["lesg"] = {8922, 65024},  
4649 ["lesseqgtr"] = 8922,  
4650 ["GreaterEqualLess"] = 8923,  
4651 ["gel"] = 8923,  
4652 ["gesl"] = {8923, 65024},  
4653 ["gtreqless"] = 8923,  
4654 ["cuepr"] = 8926,  
4655 ["curlyeqprec"] = 8926,  
4656 ["cuesc"] = 8927,  
4657 ["curlyeqsucc"] = 8927,  
4658 ["NotPrecedesSlantEqual"] = 8928,  
4659 ["nprcue"] = 8928,  
4660 ["NotSucceedsSlantEqual"] = 8929,

4661 ["nsccue"] = 8929,  
4662 ["NotSquareSubsetEqual"] = 8930,  
4663 ["nsqsube"] = 8930,  
4664 ["NotSquareSupersetEqual"] = 8931,  
4665 ["nsqsupe"] = 8931,  
4666 ["lnsim"] = 8934,  
4667 ["gnsim"] = 8935,  
4668 ["precnsim"] = 8936,  
4669 ["prnsim"] = 8936,  
4670 ["scnsim"] = 8937,  
4671 ["succnsim"] = 8937,  
4672 ["NotLeftTriangle"] = 8938,  
4673 ["nltri"] = 8938,  
4674 ["ntriangleleft"] = 8938,  
4675 ["NotRightTriangle"] = 8939,  
4676 ["nrtri"] = 8939,  
4677 ["ntriangleright"] = 8939,  
4678 ["NotLeftTriangleEqual"] = 8940,  
4679 ["nltrie"] = 8940,  
4680 ["ntrianglelefteq"] = 8940,  
4681 ["NotRightTriangleEqual"] = 8941,  
4682 ["nrtrie"] = 8941,  
4683 ["ntrianglerighteq"] = 8941,  
4684 ["vellip"] = 8942,  
4685 ["ctdot"] = 8943,  
4686 ["utdot"] = 8944,  
4687 ["dtdot"] = 8945,  
4688 ["disin"] = 8946,  
4689 ["isinsv"] = 8947,  
4690 ["isins"] = 8948,  
4691 ["isindot"] = 8949,  
4692 ["notinvc"] = {8949, 824},  
4693 ["notinvb"] = 8950,  
4694 ["notinE"] = 8951,  
4695 ["notinE"] = {8953, 824},  
4696 ["nisd"] = 8954,  
4697 ["xnis"] = 8955,  
4698 ["nis"] = 8956,  
4700 ["notnivc"] = 8957,  
4701 ["notnivb"] = 8958,  
4702 ["barwed"] = 8965,  
4703 ["barwedge"] = 8965,  
4704 ["Barwed"] = 8966,  
4705 ["doublebarwedge"] = 8966,  
4706 ["LeftCeiling"] = 8968,  
4707 ["lceil"] = 8968,



4708 ["RightCeiling"] = 8969,  
4709 ["rceil"] = 8969,  
4710 ["LeftFloor"] = 8970,  
4711 ["lfloor"] = 8970,  
4712 ["RightFloor"] = 8971,  
4713 ["rfloor"] = 8971,  
4714 ["drcrop"] = 8972,  
4715 ["dlcrop"] = 8973,  
4716 ["urcrop"] = 8974,  
4717 ["ulcrop"] = 8975,  
4718 ["bnot"] = 8976,  
4719 ["proflin"] = 8978,  
4720 ["profsurf"] = 8979,  
4721 ["telrec"] = 8981,  
4722 ["target"] = 8982,  
4723 ["ulcorn"] = 8988,  
4724 ["ulcorner"] = 8988,  
4725 ["urcorn"] = 8989,  
4726 ["urcorner"] = 8989,  
4727 ["dlcorn"] = 8990,  
4728 ["llcorner"] = 8990,  
4729 ["drcorn"] = 8991,  
4730 ["lrcorn"] = 8991,  
4731 ["frown"] = 8994,  
4732 ["sfrown"] = 8994,  
4733 ["smile"] = 8995,  
4734 ["ssmile"] = 8995,  
4735 ["cylcty"] = 9005,  
4736 ["profalar"] = 9006,  
4737 ["topbot"] = 9014,  
4738 ["ovbar"] = 9021,  
4739 ["solbar"] = 9023,  
4740 ["angzarr"] = 9084,  
4741 ["lmoust"] = 9136,  
4742 ["lmoustache"] = 9136,  
4743 ["rmoust"] = 9137,  
4744 ["rmoustache"] = 9137,  
4745 ["OverBracket"] = 9140,  
4746 ["tbrk"] = 9140,  
4747 ["UnderBracket"] = 9141,  
4748 ["bbrk"] = 9141,  
4749 ["bbrktbrk"] = 9142,  
4750 ["OverParenthesis"] = 9180,  
4751 ["UnderParenthesis"] = 9181,  
4752 ["OverBrace"] = 9182,  
4753 ["UnderBrace"] = 9183,  
4754 ["trpezium"] = 9186,

4755 ["elinters"] = 9191,  
4756 ["blank"] = 9251,  
4757 ["circledS"] = 9416,  
4758 ["oS"] = 9416,  
4759 ["HorizontalLine"] = 9472,  
4760 ["boxh"] = 9472,  
4761 ["boxv"] = 9474,  
4762 ["boxdr"] = 9484,  
4763 ["boxdl"] = 9488,  
4764 ["boxur"] = 9492,  
4765 ["boxul"] = 9496,  
4766 ["boxvr"] = 9500,  
4767 ["boxvl"] = 9508,  
4768 ["boxhd"] = 9516,  
4769 ["boxhu"] = 9524,  
4770 ["boxvh"] = 9532,  
4771 ["boxH"] = 9552,  
4772 ["boxV"] = 9553,  
4773 ["boxdR"] = 9554,  
4774 ["boxDr"] = 9555,  
4775 ["boxDR"] = 9556,  
4776 ["boxdL"] = 9557,  
4777 ["boxDL"] = 9558,  
4778 ["boxDL"] = 9559,  
4779 ["boxuR"] = 9560,  
4780 ["boxUr"] = 9561,  
4781 ["boxUR"] = 9562,  
4782 ["boxuL"] = 9563,  
4783 ["boxUL"] = 9564,  
4784 ["boxUL"] = 9565,  
4785 ["boxvR"] = 9566,  
4786 ["boxVr"] = 9567,  
4787 ["boxVR"] = 9568,  
4788 ["boxvL"] = 9569,  
4789 ["boxVl"] = 9570,  
4790 ["boxVL"] = 9571,  
4791 ["boxHd"] = 9572,  
4792 ["boxhD"] = 9573,  
4793 ["boxHD"] = 9574,  
4794 ["boxHu"] = 9575,  
4795 ["boxhU"] = 9576,  
4796 ["boxHU"] = 9577,  
4797 ["boxvH"] = 9578,  
4798 ["boxVh"] = 9579,  
4799 ["boxVH"] = 9580,  
4800 ["uhblk"] = 9600,  
4801 ["lhblk"] = 9604,

```

4802 ["block"] = 9608,
4803 ["blk14"] = 9617,
4804 ["blk12"] = 9618,
4805 ["blk34"] = 9619,
4806 ["Square"] = 9633,
4807 ["squ"] = 9633,
4808 ["square"] = 9633,
4809 ["FilledVerySmallSquare"] = 9642,
4810 ["blacksquare"] = 9642,
4811 ["squarf"] = 9642,
4812 ["squf"] = 9642,
4813 ["EmptyVerySmallSquare"] = 9643,
4814 ["rect"] = 9645,
4815 ["marker"] = 9646,
4816 ["fltns"] = 9649,
4817 ["bigtriangleup"] = 9651,
4818 ["xutri"] = 9651,
4819 ["blacktriangle"] = 9652,
4820 ["utrif"] = 9652,
4821 ["triangle"] = 9653,
4822 ["utri"] = 9653,
4823 ["blacktriangleright"] = 9656,
4824 ["rtrif"] = 9656,
4825 ["rtri"] = 9657,
4826 ["triangleright"] = 9657,
4827 ["bigtriangledown"] = 9661,
4828 ["xdtri"] = 9661,
4829 ["blacktriangledown"] = 9662,
4830 ["dtrif"] = 9662,
4831 ["dtri"] = 9663,
4832 ["triangledown"] = 9663,
4833 ["blacktriangleleft"] = 9666,
4834 ["ltrif"] = 9666,
4835 ["ltri"] = 9667,
4836 ["triangleleft"] = 9667,
4837 ["loz"] = 9674,
4838 ["lozenge"] = 9674,
4839 ["cir"] = 9675,
4840 ["tridot"] = 9708,
4841 ["bigcirc"] = 9711,
4842 ["xcirc"] = 9711,
4843 ["ultri"] = 9720,
4844 ["urtri"] = 9721,
4845 ["lltri"] = 9722,
4846 ["EmptySmallSquare"] = 9723,
4847 ["FilledSmallSquare"] = 9724,
4848 ["bigstar"] = 9733,

```

4849 ["starf"] = 9733,  
 4850 ["star"] = 9734,  
 4851 ["phone"] = 9742,  
 4852 ["female"] = 9792,  
 4853 ["male"] = 9794,  
 4854 ["spades"] = 9824,  
 4855 ["spadesuit"] = 9824,  
 4856 ["clubs"] = 9827,  
 4857 ["clubsuit"] = 9827,  
 4858 ["hearts"] = 9829,  
 4859 ["heartsuit"] = 9829,  
 4860 ["diamondsuit"] = 9830,  
 4861 ["diams"] = 9830,  
 4862 ["sung"] = 9834,  
 4863 ["flat"] = 9837,  
 4864 ["natur"] = 9838,  
 4865 ["natural"] = 9838,  
 4866 ["sharp"] = 9839,  
 4867 ["check"] = 10003,  
 4868 ["checkmark"] = 10003,  
 4869 ["cross"] = 10007,  
 4870 ["malt"] = 10016,  
 4871 ["maltese"] = 10016,  
 4872 ["sext"] = 10038,  
 4873 ["VerticalSeparator"] = 10072,  
 4874 ["lbrk"] = 10098,  
 4875 ["rbrk"] = 10099,  
 4876 ["bsolhsub"] = 10184,  
 4877 ["suphsol"] = 10185,  
 4878 ["LeftDoubleBracket"] = 10214,  
 4879 ["lobrk"] = 10214,  
 4880 ["RightDoubleBracket"] = 10215,  
 4881 ["robrk"] = 10215,  
 4882 ["LeftAngleBracket"] = 10216,  
 4883 ["lang"] = 10216,  
 4884 ["langle"] = 10216,  
 4885 ["RightAngleBracket"] = 10217,  
 4886 ["rang"] = 10217,  
 4887 ["rangle"] = 10217,  
 4888 ["Lang"] = 10218,  
 4889 ["Rang"] = 10219,  
 4890 ["loang"] = 10220,  
 4891 ["roang"] = 10221,  
 4892 ["LongLeftArrow"] = 10229,  
 4893 ["longleftarrow"] = 10229,  
 4894 ["xlarr"] = 10229,  
 4895 ["LongRightArrow"] = 10230,

4896 ["longrightarrow"] = 10230,  
 4897 ["xrarr"] = 10230,  
 4898 ["LongLeftRightArrow"] = 10231,  
 4899 ["longleftrightarrow"] = 10231,  
 4900 ["xharr"] = 10231,  
 4901 ["DoubleLongLeftArrow"] = 10232,  
 4902 ["Longleftarrow"] = 10232,  
 4903 ["xlArr"] = 10232,  
 4904 ["DoubleLongRightArrow"] = 10233,  
 4905 ["Longrightarrow"] = 10233,  
 4906 ["xrArr"] = 10233,  
 4907 ["DoubleLongLeftRightArrow"] = 10234,  
 4908 ["Longlefttrirightarrow"] = 10234,  
 4909 ["xhArr"] = 10234,  
 4910 ["longmapsto"] = 10236,  
 4911 ["xmap"] = 10236,  
 4912 ["dzigrarr"] = 10239,  
 4913 ["nvlArr"] = 10498,  
 4914 ["nvrArr"] = 10499,  
 4915 ["nvHarr"] = 10500,  
 4916 ["Map"] = 10501,  
 4917 ["lbarr"] = 10508,  
 4918 ["bkarow"] = 10509,  
 4919 ["rbarr"] = 10509,  
 4920 ["lBarr"] = 10510,  
 4921 ["dbkarow"] = 10511,  
 4922 ["rBarr"] = 10511,  
 4923 ["RBarr"] = 10512,  
 4924 ["drbkarow"] = 10512,  
 4925 ["DDotrahd"] = 10513,  
 4926 ["UpArrowBar"] = 10514,  
 4927 ["DownArrowBar"] = 10515,  
 4928 ["Rarrtl"] = 10518,  
 4929 ["latail"] = 10521,  
 4930 ["ratail"] = 10522,  
 4931 ["lAtail"] = 10523,  
 4932 ["rAtail"] = 10524,  
 4933 ["larrfs"] = 10525,  
 4934 ["rarrfs"] = 10526,  
 4935 ["larrbfs"] = 10527,  
 4936 ["rarrbfs"] = 10528,  
 4937 ["nwarhk"] = 10531,  
 4938 ["nearhk"] = 10532,  
 4939 ["hksearrow"] = 10533,  
 4940 ["searhk"] = 10533,  
 4941 ["hkswarrow"] = 10534,  
 4942 ["swarhk"] = 10534,

4943 ["nwnear"] = 10535,  
4944 ["nesear"] = 10536,  
4945 ["toea"] = 10536,  
4946 ["seswar"] = 10537,  
4947 ["tosa"] = 10537,  
4948 ["swnwar"] = 10538,  
4949 ["nrarrc"] = {10547, 824},  
4950 ["rarrc"] = 10547,  
4951 ["cudarr"] = 10549,  
4952 ["ldca"] = 10550,  
4953 ["rdca"] = 10551,  
4954 ["cudarrl"] = 10552,  
4955 ["larrpl"] = 10553,  
4956 ["curarrm"] = 10556,  
4957 ["cularrp"] = 10557,  
4958 ["rarrpl"] = 10565,  
4959 ["harrcir"] = 10568,  
4960 ["Uarrocir"] = 10569,  
4961 ["lurdshar"] = 10570,  
4962 ["ldrushar"] = 10571,  
4963 ["LeftRightVector"] = 10574,  
4964 ["RightUpDownVector"] = 10575,  
4965 ["DownLeftRightVector"] = 10576,  
4966 ["LeftUpDownVector"] = 10577,  
4967 ["LeftVectorBar"] = 10578,  
4968 ["RightVectorBar"] = 10579,  
4969 ["RightUpVectorBar"] = 10580,  
4970 ["RightDownVectorBar"] = 10581,  
4971 ["DownLeftVectorBar"] = 10582,  
4972 ["DownRightVectorBar"] = 10583,  
4973 ["LeftUpVectorBar"] = 10584,  
4974 ["LeftDownVectorBar"] = 10585,  
4975 ["LeftTeeVector"] = 10586,  
4976 ["RightTeeVector"] = 10587,  
4977 ["RightUpTeeVector"] = 10588,  
4978 ["RightDownTeeVector"] = 10589,  
4979 ["DownLeftTeeVector"] = 10590,  
4980 ["DownRightTeeVector"] = 10591,  
4981 ["LeftUpTeeVector"] = 10592,  
4982 ["LeftDownTeeVector"] = 10593,  
4983 ["lHar"] = 10594,  
4984 ["uHar"] = 10595,  
4985 ["rHar"] = 10596,  
4986 ["dHar"] = 10597,  
4987 ["luruhar"] = 10598,  
4988 ["ldrdhar"] = 10599,  
4989 ["ruluhar"] = 10600,

4990 ["rdldhar"] = 10601,  
 4991 ["lharul"] = 10602,  
 4992 ["llhard"] = 10603,  
 4993 ["rharul"] = 10604,  
 4994 ["lrhard"] = 10605,  
 4995 ["UpEquilibrium"] = 10606,  
 4996 ["udhar"] = 10606,  
 4997 ["ReverseUpEquilibrium"] = 10607,  
 4998 ["duhar"] = 10607,  
 4999 ["RoundImplies"] = 10608,  
 5000 ["erarr"] = 10609,  
 5001 ["simrarr"] = 10610,  
 5002 ["larrsim"] = 10611,  
 5003 ["rarrsim"] = 10612,  
 5004 ["rarrap"] = 10613,  
 5005 ["ltlarr"] = 10614,  
 5006 ["gtrarr"] = 10616,  
 5007 ["subrarr"] = 10617,  
 5008 ["suplarr"] = 10619,  
 5009 ["lfisht"] = 10620,  
 5010 ["rfisht"] = 10621,  
 5011 ["ufisht"] = 10622,  
 5012 ["dfisht"] = 10623,  
 5013 ["lopar"] = 10629,  
 5014 ["ropar"] = 10630,  
 5015 ["lbrke"] = 10635,  
 5016 ["rbrke"] = 10636,  
 5017 ["lbrkslu"] = 10637,  
 5018 ["rbrksld"] = 10638,  
 5019 ["lbrksld"] = 10639,  
 5020 ["rbrkslu"] = 10640,  
 5021 ["langd"] = 10641,  
 5022 ["rangd"] = 10642,  
 5023 ["lparlt"] = 10643,  
 5024 ["rpargt"] = 10644,  
 5025 ["gtlPar"] = 10645,  
 5026 ["ltrPar"] = 10646,  
 5027 ["vzigzag"] = 10650,  
 5028 ["vangrt"] = 10652,  
 5029 ["angrtvbd"] = 10653,  
 5030 ["ange"] = 10660,  
 5031 ["range"] = 10661,  
 5032 ["dwangle"] = 10662,  
 5033 ["uwangle"] = 10663,  
 5034 ["angmsdaa"] = 10664,  
 5035 ["angmsdab"] = 10665,  
 5036 ["angmsdac"] = 10666,

5037 ["angmsdad"] = 10667,  
5038 ["angmsdae"] = 10668,  
5039 ["angmsdaf"] = 10669,  
5040 ["angmsdag"] = 10670,  
5041 ["angmsdah"] = 10671,  
5042 ["bemptyv"] = 10672,  
5043 ["demptyv"] = 10673,  
5044 ["cemptyv"] = 10674,  
5045 ["raemtyv"] = 10675,  
5046 ["laemtyv"] = 10676,  
5047 ["ohbar"] = 10677,  
5048 ["omid"] = 10678,  
5049 ["opar"] = 10679,  
5050 ["operp"] = 10681,  
5051 ["olcross"] = 10683,  
5052 ["odsold"] = 10684,  
5053 ["olcir"] = 10686,  
5054 ["ofcir"] = 10687,  
5055 ["olt"] = 10688,  
5056 ["ogt"] = 10689,  
5057 ["cirscir"] = 10690,  
5058 ["cirE"] = 10691,  
5059 ["solb"] = 10692,  
5060 ["bsolb"] = 10693,  
5061 ["boxbox"] = 10697,  
5062 ["trisb"] = 10701,  
5063 ["rtriltri"] = 10702,  
5064 ["LeftTriangleBar"] = 10703,  
5065 ["NotLeftTriangleBar"] = {10703, 824},  
5066 ["NotRightTriangleBar"] = {10704, 824},  
5067 ["RightTriangleBar"] = 10704,  
5068 ["iinfin"] = 10716,  
5069 ["infintie"] = 10717,  
5070 ["nvinfin"] = 10718,  
5071 ["eparsl"] = 10723,  
5072 ["smeparsl"] = 10724,  
5073 ["eqvparsl"] = 10725,  
5074 ["blacklozenge"] = 10731,  
5075 ["lozf"] = 10731,  
5076 ["RuleDelayed"] = 10740,  
5077 ["dsol"] = 10742,  
5078 ["bigodot"] = 10752,  
5079 ["xodot"] = 10752,  
5080 ["bigoplus"] = 10753,  
5081 ["xoplus"] = 10753,  
5082 ["bigotimes"] = 10754,  
5083 ["xotime"] = 10754,



5084 ["biguplus"] = 10756,  
5085 ["xuplus"] = 10756,  
5086 ["bigsqcup"] = 10758,  
5087 ["xscup"] = 10758,  
5088 ["iiiint"] = 10764,  
5089 ["qint"] = 10764,  
5090 ["fpartint"] = 10765,  
5091 ["cirfnint"] = 10768,  
5092 ["awint"] = 10769,  
5093 ["rppolint"] = 10770,  
5094 ["scpolint"] = 10771,  
5095 ["npolint"] = 10772,  
5096 ["pointint"] = 10773,  
5097 ["quatint"] = 10774,  
5098 ["intlarhk"] = 10775,  
5099 ["pluscir"] = 10786,  
5100 ["plusacir"] = 10787,  
5101 ["simplus"] = 10788,  
5102 ["plusdu"] = 10789,  
5103 ["plussim"] = 10790,  
5104 ["plustwo"] = 10791,  
5105 ["mcomma"] = 10793,  
5106 ["minusdu"] = 10794,  
5107 ["loplus"] = 10797,  
5108 ["roplus"] = 10798,  
5109 ["Cross"] = 10799,  
5110 ["timesd"] = 10800,  
5111 ["timesbar"] = 10801,  
5112 ["smashp"] = 10803,  
5113 ["lotimes"] = 10804,  
5114 ["rotimes"] = 10805,  
5115 ["otimesas"] = 10806,  
5116 ["Otimes"] = 10807,  
5117 ["odiv"] = 10808,  
5118 ["triplus"] = 10809,  
5119 ["triminus"] = 10810,  
5120 ["tritime"] = 10811,  
5121 ["intprod"] = 10812,  
5122 ["iproduct"] = 10812,  
5123 ["amalg"] = 10815,  
5124 ["capdot"] = 10816,  
5125 ["ncup"] = 10818,  
5126 ["ncap"] = 10819,  
5127 ["capand"] = 10820,  
5128 ["cupor"] = 10821,  
5129 ["cupcap"] = 10822,  
5130 ["capcup"] = 10823,

5131 ["cupbrcap"] = 10824,  
5132 ["capbrcup"] = 10825,  
5133 ["cupcup"] = 10826,  
5134 ["capcap"] = 10827,  
5135 ["ccups"] = 10828,  
5136 ["ccaps"] = 10829,  
5137 ["ccupssm"] = 10832,  
5138 ["And"] = 10835,  
5139 ["Or"] = 10836,  
5140 ["andand"] = 10837,  
5141 ["oror"] = 10838,  
5142 ["orslope"] = 10839,  
5143 ["andslope"] = 10840,  
5144 ["andv"] = 10842,  
5145 ["orv"] = 10843,  
5146 ["andd"] = 10844,  
5147 ["ord"] = 10845,  
5148 ["wedbar"] = 10847,  
5149 ["sdote"] = 10854,  
5150 ["simdot"] = 10858,  
5151 ["congdote"] = 10861,  
5152 ["ncongdote"] = {10861, 824},  
5153 ["easter"] = 10862,  
5154 ["apacir"] = 10863,  
5155 ["apE"] = 10864,  
5156 ["napE"] = {10864, 824},  
5157 ["eplus"] = 10865,  
5158 ["pluse"] = 10866,  
5159 ["Esim"] = 10867,  
5160 ["Colone"] = 10868,  
5161 ["Equal"] = 10869,  
5162 ["ddotseq"] = 10871,  
5163 ["eDDot"] = 10871,  
5164 ["equivDD"] = 10872,  
5165 ["ltcir"] = 10873,  
5166 ["gtcir"] = 10874,  
5167 ["ltquest"] = 10875,  
5168 ["gtquest"] = 10876,  
5169 ["LessSlantEqual"] = 10877,  
5170 ["NotLessSlantEqual"] = {10877, 824},  
5171 ["leqslant"] = 10877,  
5172 ["les"] = 10877,  
5173 ["nleqslant"] = {10877, 824},  
5174 ["nles"] = {10877, 824},  
5175 ["GreaterSlantEqual"] = 10878,  
5176 ["NotGreaterSlantEqual"] = {10878, 824},  
5177 ["geqslant"] = 10878,

5178 ["ges"] = 10878,  
5179 ["ngeqslant"] = {10878, 824},  
5180 ["nges"] = {10878, 824},  
5181 ["lesdot"] = 10879,  
5182 ["gesdot"] = 10880,  
5183 ["lesdoto"] = 10881,  
5184 ["gesdoto"] = 10882,  
5185 ["lesdotor"] = 10883,  
5186 ["gesdoto1"] = 10884,  
5187 ["lap"] = 10885,  
5188 ["lessapprox"] = 10885,  
5189 ["gap"] = 10886,  
5190 ["gtrapprox"] = 10886,  
5191 ["lne"] = 10887,  
5192 ["lneq"] = 10887,  
5193 ["gne"] = 10888,  
5194 ["gneq"] = 10888,  
5195 ["lnap"] = 10889,  
5196 ["lnapprox"] = 10889,  
5197 ["gnap"] = 10890,  
5198 ["gnapprox"] = 10890,  
5199 ["lEg"] = 10891,  
5200 ["lesseqgtr"] = 10891,  
5201 ["gEl"] = 10892,  
5202 ["gtreqqless"] = 10892,  
5203 ["lsime"] = 10893,  
5204 ["gsime"] = 10894,  
5205 ["lsimg"] = 10895,  
5206 ["gsiml"] = 10896,  
5207 ["lgE"] = 10897,  
5208 ["glE"] = 10898,  
5209 ["lesges"] = 10899,  
5210 ["gesles"] = 10900,  
5211 ["els"] = 10901,  
5212 ["eqslantless"] = 10901,  
5213 ["egs"] = 10902,  
5214 ["eqslantgtr"] = 10902,  
5215 ["elsdot"] = 10903,  
5216 ["egsdot"] = 10904,  
5217 ["el"] = 10905,  
5218 ["eg"] = 10906,  
5219 ["siml"] = 10909,  
5220 ["sing"] = 10910,  
5221 ["simLE"] = 10911,  
5222 ["singE"] = 10912,  
5223 ["LessLess"] = 10913,  
5224 ["NotNestedLessLess"] = {10913, 824},

```

5225 ["GreaterGreater"] = 10914,
5226 ["NotNestedGreaterGreater"] = {10914, 824},
5227 ["glj"] = 10916,
5228 ["gla"] = 10917,
5229 ["ltcc"] = 10918,
5230 ["gtcc"] = 10919,
5231 ["lescc"] = 10920,
5232 ["gescc"] = 10921,
5233 ["smt"] = 10922,
5234 ["lat"] = 10923,
5235 ["smte"] = 10924,
5236 ["smtes"] = {10924, 65024},
5237 ["late"] = 10925,
5238 ["lates"] = {10925, 65024},
5239 ["bumpE"] = 10926,
5240 ["NotPrecedesEqual"] = {10927, 824},
5241 ["PrecedesEqual"] = 10927,
5242 ["npre"] = {10927, 824},
5243 ["npreceq"] = {10927, 824},
5244 ["pre"] = 10927,
5245 ["preceq"] = 10927,
5246 ["NotSucceedsEqual"] = {10928, 824},
5247 ["SucceedsEqual"] = 10928,
5248 ["nsce"] = {10928, 824},
5249 ["nsucceq"] = {10928, 824},
5250 ["sce"] = 10928,
5251 ["succeq"] = 10928,
5252 ["prE"] = 10931,
5253 ["scE"] = 10932,
5254 ["precneqq"] = 10933,
5255 ["prnE"] = 10933,
5256 ["scnE"] = 10934,
5257 ["succneqq"] = 10934,
5258 ["prap"] = 10935,
5259 ["precapprox"] = 10935,
5260 ["scap"] = 10936,
5261 ["succapprox"] = 10936,
5262 ["precnapprox"] = 10937,
5263 ["prnap"] = 10937,
5264 ["scnap"] = 10938,
5265 ["succnapprox"] = 10938,
5266 ["Pr"] = 10939,
5267 ["Sc"] = 10940,
5268 ["subdot"] = 10941,
5269 ["supdot"] = 10942,
5270 ["subplus"] = 10943,
5271 ["supplus"] = 10944,

```

5272 ["submult"] = 10945,  
5273 ["supmult"] = 10946,  
5274 ["subedot"] = 10947,  
5275 ["supedot"] = 10948,  
5276 ["nsubE"] = {10949, 824},  
5277 ["nsubseteqq"] = {10949, 824},  
5278 ["subE"] = 10949,  
5279 ["subseteqq"] = 10949,  
5280 ["nsupE"] = {10950, 824},  
5281 ["nsupseteqq"] = {10950, 824},  
5282 ["supE"] = 10950,  
5283 ["supseteqq"] = 10950,  
5284 ["subsim"] = 10951,  
5285 ["supsim"] = 10952,  
5286 ["subnE"] = 10955,  
5287 ["subsetneqq"] = 10955,  
5288 ["varsubsetneqq"] = {10955, 65024},  
5289 ["vsubnE"] = {10955, 65024},  
5290 ["supnE"] = 10956,  
5291 ["supsetneqq"] = 10956,  
5292 ["varsupsetneqq"] = {10956, 65024},  
5293 ["vsupnE"] = {10956, 65024},  
5294 ["csub"] = 10959,  
5295 ["csup"] = 10960,  
5296 ["csube"] = 10961,  
5297 ["csupe"] = 10962,  
5298 ["subsup"] = 10963,  
5299 ["supsub"] = 10964,  
5300 ["subsub"] = 10965,  
5301 ["supsup"] = 10966,  
5302 ["suphsub"] = 10967,  
5303 ["supdsub"] = 10968,  
5304 ["forkv"] = 10969,  
5305 ["topfork"] = 10970,  
5306 ["mlcp"] = 10971,  
5307 ["Dashv"] = 10980,  
5308 ["DoubleLeftTee"] = 10980,  
5309 ["Vdashl"] = 10982,  
5310 ["Barv"] = 10983,  
5311 ["vBar"] = 10984,  
5312 ["vBarv"] = 10985,  
5313 ["Vbar"] = 10987,  
5314 ["Not"] = 10988,  
5315 ["bNot"] = 10989,  
5316 ["rnmid"] = 10990,  
5317 ["cirmid"] = 10991,  
5318 ["midcir"] = 10992,

5319 ["topcir"] = 10993,  
5320 ["nhpar"] = 10994,  
5321 ["parsim"] = 10995,  
5322 ["nparsl"] = {11005, 8421},  
5323 ["parsl"] = 11005,  
5324 ["fflig"] = 64256,  
5325 ["filig"] = 64257,  
5326 ["fllig"] = 64258,  
5327 ["ffilig"] = 64259,  
5328 ["ffllig"] = 64260,  
5329 ["Ascr"] = 119964,  
5330 ["Cscr"] = 119966,  
5331 ["Dscr"] = 119967,  
5332 ["Gscr"] = 119970,  
5333 ["Jscr"] = 119973,  
5334 ["Kscr"] = 119974,  
5335 ["Nscr"] = 119977,  
5336 ["Oscr"] = 119978,  
5337 ["Pscr"] = 119979,  
5338 ["Qscr"] = 119980,  
5339 ["Sscr"] = 119982,  
5340 ["Tscr"] = 119983,  
5341 ["Uscr"] = 119984,  
5342 ["Vscr"] = 119985,  
5343 ["Wscr"] = 119986,  
5344 ["Xscr"] = 119987,  
5345 ["Yscr"] = 119988,  
5346 ["Zscr"] = 119989,  
5347 ["ascr"] = 119990,  
5348 ["bscr"] = 119991,  
5349 ["cscr"] = 119992,  
5350 ["dscr"] = 119993,  
5351 ["fscr"] = 119995,  
5352 ["hscr"] = 119997,  
5353 ["iscr"] = 119998,  
5354 ["jscr"] = 119999,  
5355 ["kscr"] = 120000,  
5356 ["lscr"] = 120001,  
5357 ["mscr"] = 120002,  
5358 ["nscr"] = 120003,  
5359 ["pscr"] = 120005,  
5360 ["qscr"] = 120006,  
5361 ["rscr"] = 120007,  
5362 ["sscr"] = 120008,  
5363 ["tscr"] = 120009,  
5364 ["uscr"] = 120010,  
5365 ["vscr"] = 120011,

5366 ["wscr"] = 120012,  
5367 ["xscr"] = 120013,  
5368 ["yscr"] = 120014,  
5369 ["zscr"] = 120015,  
5370 ["Afr"] = 120068,  
5371 ["Bfr"] = 120069,  
5372 ["Dfr"] = 120071,  
5373 ["Efr"] = 120072,  
5374 ["Ffr"] = 120073,  
5375 ["Gfr"] = 120074,  
5376 ["Jfr"] = 120077,  
5377 ["Kfr"] = 120078,  
5378 ["Lfr"] = 120079,  
5379 ["Mfr"] = 120080,  
5380 ["Nfr"] = 120081,  
5381 ["Ofr"] = 120082,  
5382 ["Pfr"] = 120083,  
5383 ["Qfr"] = 120084,  
5384 ["Sfr"] = 120086,  
5385 ["Tfr"] = 120087,  
5386 ["Ufr"] = 120088,  
5387 ["Vfr"] = 120089,  
5388 ["Wfr"] = 120090,  
5389 ["Xfr"] = 120091,  
5390 ["Yfr"] = 120092,  
5391 ["afr"] = 120094,  
5392 ["bfr"] = 120095,  
5393 ["cfr"] = 120096,  
5394 ["dfr"] = 120097,  
5395 ["efr"] = 120098,  
5396 ["ffr"] = 120099,  
5397 ["gfr"] = 120100,  
5398 ["hfr"] = 120101,  
5399 ["ifr"] = 120102,  
5400 ["jfr"] = 120103,  
5401 ["kfr"] = 120104,  
5402 ["lfr"] = 120105,  
5403 ["mfr"] = 120106,  
5404 ["nfr"] = 120107,  
5405 ["ofr"] = 120108,  
5406 ["pfr"] = 120109,  
5407 ["qfr"] = 120110,  
5408 ["rfr"] = 120111,  
5409 ["sfr"] = 120112,  
5410 ["tfr"] = 120113,  
5411 ["ufr"] = 120114,  
5412 ["vfr"] = 120115,

5413 ["wfr"] = 120116,  
5414 ["xfr"] = 120117,  
5415 ["yfr"] = 120118,  
5416 ["zfr"] = 120119,  
5417 ["Aopf"] = 120120,  
5418 ["Bopf"] = 120121,  
5419 ["Dopf"] = 120123,  
5420 ["Eopf"] = 120124,  
5421 ["Fopf"] = 120125,  
5422 ["Gopf"] = 120126,  
5423 ["Iopf"] = 120128,  
5424 ["Jopf"] = 120129,  
5425 ["Kopf"] = 120130,  
5426 ["Lopf"] = 120131,  
5427 ["Mopf"] = 120132,  
5428 ["Oopf"] = 120134,  
5429 ["Sopf"] = 120138,  
5430 ["Topf"] = 120139,  
5431 ["Uopf"] = 120140,  
5432 ["Vopf"] = 120141,  
5433 ["Wopf"] = 120142,  
5434 ["Xopf"] = 120143,  
5435 ["Yopf"] = 120144,  
5436 ["aopf"] = 120146,  
5437 ["bopf"] = 120147,  
5438 ["copf"] = 120148,  
5439 ["dopf"] = 120149,  
5440 ["eopf"] = 120150,  
5441 ["fopf"] = 120151,  
5442 ["gopf"] = 120152,  
5443 ["hopf"] = 120153,  
5444 ["iopf"] = 120154,  
5445 ["jopf"] = 120155,  
5446 ["kopf"] = 120156,  
5447 ["lopf"] = 120157,  
5448 ["mopf"] = 120158,  
5449 ["nopf"] = 120159,  
5450 ["oopf"] = 120160,  
5451 ["popf"] = 120161,  
5452 ["qopf"] = 120162,  
5453 ["ropf"] = 120163,  
5454 ["sopf"] = 120164,  
5455 ["topf"] = 120165,  
5456 ["uopf"] = 120166,  
5457 ["vopf"] = 120167,  
5458 ["wopf"] = 120168,  
5459 ["xopf"] = 120169,



```

5460 ["yopf"] = 120170,
5461 ["zopf"] = 120171,
5462 }

```

Given a string `s` of decimal digits, the `entities.dec_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

5463 function entities.dec_entity(s)
5464 local n = tonumber(s)
5465 if n == nil then
5466 return "&#" .. s .. ";" -- fallback for unknown entities
5467 end
5468 return unicode.utf8.char(n)
5469 end

```

Given a string `s` of hexadecimal digits, the `entities.hex_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

5470 function entities.hex_entity(s)
5471 local n = tonumber("0x"..s)
5472 if n == nil then
5473 return "&#x" .. s .. ";" -- fallback for unknown entities
5474 end
5475 return unicode.utf8.char(n)
5476 end

```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```

5477 function entities.hex_entity_with_x_char(x, s)
5478 local n = tonumber("0x"..s)
5479 if n == nil then
5480 return "&#" .. x .. s .. ";" -- fallback for unknown entities
5481 end
5482 return unicode.utf8.char(n)
5483 end

```

Given a character entity name `s` (like `ouml`), the `entities.char_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

5484 function entities.char_entity(s)
5485 local code_points = character_entities[s]
5486 if code_points == nil then
5487 return "&" .. s .. ";"
5488 end
5489 if type(code_points) ~= 'table' then
5490 code_points = {code_points}
5491 end
5492 local char_table = {}
5493 for _, code_point in ipairs(code_points) do
5494 table.insert(char_table, unicode.utf8.char(code_point))

```

```

5495 end
5496 return table.concat(char_table)
5497 end

```

### 3.1.3 Plain T<sub>E</sub>X Writer

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark Lua module.

Although not specified in the Lua interface (see Section 2.1), the `writer` object is exported, so that the curious user could easily tinker with the methods of the objects produced by the `writer.new` method described below. The user should be aware, however, that the implementation may change in a future revision.

```

5498 M.writer = {}

```

The `writer.new` method creates and returns a new T<sub>E</sub>X writer object associated with the Lua interface options (see Section 2.1.3) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `writer->member`. All member variables are immutable unless explicitly stated otherwise.

```

5499 function M.writer.new(options)
5500 local self = {}

```

Make `options` available as `writer->options`, so that it is accessible from extensions.

```

5501 self.options = options

```

Define `writer->flatten_inlines`, which indicates whether or not the writer should produce raw text rather than text in the output format for inline elements. The `writer->flatten_inlines` member variable is mutable.

```

5502 self.flatten_inlines = false

```

Parse the `slice` option and define `writer->slice_begin`, `writer->slice_end`, and `writer->is_writing`. The `writer->is_writing` member variable is mutable.

```

5503 local slice_specifiers = {}
5504 for specifier in options.slice:gmatch("[^%s]+") do
5505 table.insert(slice_specifiers, specifier)
5506 end
5507
5508 if #slice_specifiers == 2 then
5509 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5510 local slice_begin_type = self.slice_begin:sub(1, 1)

```

```

5511 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5512 self.slice_begin = "^" .. self.slice_begin
5513 end
5514 local slice_end_type = self.slice_end:sub(1, 1)
5515 if slice_end_type ~= "^" and slice_end_type ~= "$" then
5516 self.slice_end = "$" .. self.slice_end
5517 end
5518 elseif #slice_specifiers == 1 then
5519 self.slice_begin = "^" .. slice_specifiers[1]
5520 self.slice_end = "$" .. slice_specifiers[1]
5521 end
5522
5523 self.slice_begin_type = self.slice_begin:sub(1, 1)
5524 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5525 self.slice_end_type = self.slice_end:sub(1, 1)
5526 self.slice_end_identifier = self.slice_end:sub(2) or ""
5527
5528 if self.slice_begin == "^" and self.slice_end ~= "^" then
5529 self.is_writing = true
5530 else
5531 self.is_writing = false
5532 end

```

Define `writer->suffix` as the suffix of the produced cache files.

```
5533 self.suffix = ".tex"
```

Define `writer->space` as the output format of a space character.

```
5534 self.space = " "
```

Define `writer->nbsp` as the output format of a non-breaking space character.

```
5535 self.nbsp = "\\markdownRendererNbsp{}"
```

Define `writer->plain` as a function that will transform an input plain text block `s` to the output format.

```
5536 function self.plain(s)
5537 return s
5538 end

```

Define `writer->paragraph` as a function that will transform an input paragraph `s` to the output format.

```
5539 function self.paragraph(s)
5540 if not self.is_writing then return "" end
5541 return s
5542 end

```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

```
5543 function self.pack(name)
5544 return [[\input{]] .. name .. [[]\relax]]
5545 end

```

Define `writer->interblocksep` as the output format of a block element separator.

```
5546 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
5547 function self.interblocksep()
5548 if not self.is_writing then return "" end
5549 return self.interblocksep_text
5550 end
```

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

```
5551 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
5552 function self.paragraphsep()
5553 if not self.is_writing then return "" end
5554 return self.paragraphsep_text
5555 end
```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
5556 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
5557 function self.undosep()
5558 if not self.is_writing then return "" end
5559 return self.undosep_text
5560 end
```

Define `writer->soft_line_break` as the output format of a soft line break.

```
5561 self.soft_line_break = function()
5562 if self.flatten_inlines then return "\n" end
5563 return "\\markdownRendererSoftLineBreak\n{"
5564 end
```

Define `writer->hard_line_break` as the output format of a hard line break.

```
5565 self.hard_line_break = function()
5566 if self.flatten_inlines then return "\n" end
5567 return "\\markdownRendererHardLineBreak\n{"
5568 end
```

Define `writer->ellipsis` as the output format of an ellipsis.

```
5569 self.ellipsis = "\\markdownRendererEllipsis{"
```

Define `writer->thematic_break` as the output format of a thematic break.

```
5570 function self.thematic_break()
5571 if not self.is_writing then return "" end
5572 return "\\markdownRendererThematicBreak{"
5573 end
```

Define tables `writer->escaped_uri_chars` and `writer->escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

```

5574 self.escaped_uri_chars = {
5575 [{""] = "\\markdownRendererLeftBrace{}"},
5576 ["}"] = "\\markdownRendererRightBrace{}"},
5577 [{"\\"} = "\\markdownRendererBackslash{}"},
5578 }
5579 self.escaped_minimal_strings = {
5580 [{"^"} = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
5581 [{"☒"} = "\\markdownRendererTickedBox{}"},
5582 [{"◻"} = "\\markdownRendererHalfTickedBox{}"},
5583 [{"□"} = "\\markdownRendererUntickedBox{}"},
5584 [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{}"},
5585 }

```

Define table `writer->escaped_strings` containing the mapping from character strings that need to be escaped in typeset content.

```

5586 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5587 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

Define a table `writer->escaped_chars` containing the mapping from special plain  $\TeX$  characters (including the active pipe character (`|`) of `Con $\TeX$ t`) that need to be escaped in typeset content.

```

5588 self.escaped_chars = {
5589 [{""] = "\\markdownRendererLeftBrace{}"},
5590 ["}"] = "\\markdownRendererRightBrace{}"},
5591 [{"%"} = "\\markdownRendererPercentSign{}"},
5592 [{"\\"} = "\\markdownRendererBackslash{}"},
5593 [{"#"} = "\\markdownRendererHash{}"},
5594 [{"$"} = "\\markdownRendererDollarSign{}"},
5595 [{"&"} = "\\markdownRendererAmpersand{}"},
5596 [{"_"} = "\\markdownRendererUnderscore{}"},
5597 [{"^"} = "\\markdownRendererCircumflex{}"},
5598 [{"~"} = "\\markdownRendererTilde{}"},
5599 [{"|"} = "\\markdownRendererPipe{}"},
5600 [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{}"},
5601 }

```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal` tables to create the `writer->escape_typographic_text`, `writer->escape_programmatic_text`, and `writer->escape_minimal` escaper functions.

```

5602 local function create_escaper(char_escapes, string_escapes)
5603 local escape = util.escaper(char_escapes, string_escapes)
5604 return function(s)
5605 if self.flatten_inlines then return s end
5606 return escape(s)
5607 end
5608 end
5609 local escape_typographic_text = create_escaper(
5610 self.escaped_chars, self.escaped_strings)

```

```

5611 local escape_programmatic_text = create_escaper(
5612 self.escaped_uri_chars, self.escaped_minimal_strings)
5613 local escape_minimal = create_escaper(
5614 {}, self.escaped_minimal_strings)

```

Define the following semantic aliases for the escaper functions:

- `writer->escape` transforms a text string that should always be made printable.
- `writer->string` transforms a text string that should be made printable only when the `hybrid` Lua option is disabled. When `hybrid` is enabled, the text string should be kept as-is.
- `writer->math` transforms a math span.
- `writer->identifier` transforms an input programmatic identifier.
- `writer->uri` transforms an input URI.
- `writer->infostring` transforms a fence code infostring.

```

5615 self.escape = escape_typographic_text
5616 self.math = escape_minimal
5617 if options.hybrid then
5618 self.identifier = escape_minimal
5619 self.string = escape_minimal
5620 self.uri = escape_minimal
5621 self.infostring = escape_minimal
5622 else
5623 self.identifier = escape_programmatic_text
5624 self.string = escape_typographic_text
5625 self.uri = escape_programmatic_text
5626 self.infostring = escape_programmatic_text
5627 end

```

Define `writer->code` as a function that will transform an input inline code span `s` with optional attributes `attributes` to the output format.

```

5628 function self.code(s, attributes)
5629 if self.flatten_inlines then return s end
5630 local buf = {}
5631 if attributes ~= nil then
5632 table.insert(buf,
5633 "\\markdownRendererCodeSpanAttributeContextBegin\n")
5634 table.insert(buf, self.attributes(attributes))
5635 end
5636 table.insert(buf,
5637 {"\\markdownRendererCodeSpan{" , self.escape(s), "}")})
5638 if attributes ~= nil then
5639 table.insert(buf,
5640 "\\markdownRendererCodeSpanAttributeContextEnd{")
5641 end
5642 return buf
5643 end

```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, `tit` to the title of the link, and `attributes` to optional attributes.

```

5644 function self.link(lab, src, tit, attributes)
5645 if self.flatten_inlines then return lab end
5646 local buf = {}
5647 if attributes ~= nil then
5648 table.insert(buf,
5649 "\\markdownRendererLinkAttributeContextBegin\n")
5650 table.insert(buf, self.attributes(attributes))
5651 end
5652 table.insert(buf, {"\\markdownRendererLink{",lab,"}",
5653 "{",self.escape(src),"",
5654 "{",self.uri(src),"",
5655 "{",self.string(tit or ""),""}})
5656 if attributes ~= nil then
5657 table.insert(buf,
5658 "\\markdownRendererLinkAttributeContextEnd{")
5659 end
5660 return buf
5661 end

```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, `tit` to the title of the image, and `attributes` to optional attributes.

```

5662 function self.image(lab, src, tit, attributes)
5663 if self.flatten_inlines then return lab end
5664 local buf = {}
5665 if attributes ~= nil then
5666 table.insert(buf,
5667 "\\markdownRendererImageAttributeContextBegin\n")
5668 table.insert(buf, self.attributes(attributes))
5669 end
5670 table.insert(buf, {"\\markdownRendererImage{",lab,"}",
5671 "{",self.string(src),"",
5672 "{",self.uri(src),"",
5673 "{",self.string(tit or ""),""}})
5674 if attributes ~= nil then
5675 table.insert(buf,
5676 "\\markdownRendererImageAttributeContextEnd{")
5677 end
5678 return buf
5679 end

```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```

5680 function self.bulletlist(items,tight)
5681 if not self.is_writing then return "" end
5682 local buffer = {}
5683 for _,item in ipairs(items) do
5684 if item ~= "" then
5685 buffer[#buffer + 1] = self.bulletitem(item)
5686 end
5687 end
5688 local contents = util.intersperse(buffer,"\n")
5689 if tight and options.tightLists then
5690 return {"\markdownRendererUlBeginTight\n",contents,
5691 "\n\markdownRendererUlEndTight "}
5692 else
5693 return {"\markdownRendererUlBegin\n",contents,
5694 "\n\markdownRendererUlEnd "}
5695 end
5696 end

```

Define `writer->bulletitem` as a function that will transform an input bulleted list item to the output format, where `s` is the text of the list item.

```

5697 function self.bulletitem(s)
5698 return {"\markdownRendererUlItem ",s,
5699 "\markdownRendererUlItemEnd "}
5700 end

```

Define `writer->orderedlist` as a function that will transform an input ordered list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not. If the optional parameter `startnum` is present, it is the number of the first list item.

```

5701 function self.orderedlist(items,tight,startnum)
5702 if not self.is_writing then return "" end
5703 local buffer = {}
5704 local num = startnum
5705 for _,item in ipairs(items) do
5706 if item ~= "" then
5707 buffer[#buffer + 1] = self.ordereditem(item,num)
5708 end
5709 if num ~= nil and item ~= "" then
5710 num = num + 1
5711 end
5712 end
5713 local contents = util.intersperse(buffer,"\n")
5714 if tight and options.tightLists then
5715 return {"\markdownRendererOlBeginTight\n",contents,
5716 "\n\markdownRendererOlEndTight "}
5717 else
5718 return {"\markdownRendererOlBegin\n",contents,

```



```

5719 "\n\\markdownRendererOlEnd "}
5720 end
5721 end

```

Define `writer->orderitem` as a function that will transform an input ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```

5722 function self.orderitem(s,num)
5723 if num ~= nil then
5724 return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
5725 "\\markdownRendererOlItemEnd "}
5726 else
5727 return {"\\markdownRendererOlItem ",s,
5728 "\\markdownRendererOlItemEnd "}
5729 end
5730 end

```

Define `writer->inline_html_comment` as a function that will transform the contents of an inline HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

5731 function self.inline_html_comment(contents)
5732 if self.flatten_inlines then return contents end
5733 return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5734 end

```

Define `writer->inline_html_tag` as a function that will transform the contents of an opening, closing, or empty inline HTML tag to the output format, where `contents` are the contents of the HTML tag.

```

5735 function self.inline_html_tag(contents)
5736 if self.flatten_inlines then return contents end
5737 return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5738 end

```

Define `writer->block_html_element` as a function that will transform the contents of a block HTML element to the output format, where `s` are the contents of the HTML element.

```

5739 function self.block_html_element(s)
5740 if not self.is_writing then return "" end
5741 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5742 return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5743 end

```

Define `writer->emphasis` as a function that will transform an emphasized span `s` of input text to the output format.

```

5744 function self.emphasis(s)
5745 if self.flatten_inlines then return s end
5746 return {"\\markdownRendererEmphasis{" ,s,"}"}
5747 end

```

Define `writer->checkbox` as a function that will transform a number `f` to the output format.

```
5748 function self.checkbox(f)
5749 if f == 1.0 then
5750 return "☒ "
5751 elseif f == 0.0 then
5752 return "☐ "
5753 else
5754 return "◻ "
5755 end
5756 end
```

Define `writer->strong` as a function that will transform a strongly emphasized span `s` of input text to the output format.

```
5757 function self.strong(s)
5758 if self.flatten_inlines then return s end
5759 return {"\\markdownRendererStrongEmphasis{" ,s,""} }
5760 end
```

Define `writer->blockquote` as a function that will transform an input block quote `s` to the output format.

```
5761 function self.blockquote(s)
5762 if not self.is_writing then return "" end
5763 return {"\\markdownRendererBlockQuoteBegin\\n",s,
5764 "\\n\\markdownRendererBlockQuoteEnd "}
5765 end
```

Define `writer->verbatim` as a function that will transform an input code block `s` to the output format.

```
5766 function self.verbatim(s)
5767 if not self.is_writing then return "" end
5768 s = s:gsub("\\n$", "")
5769 local name = util.cache_verbatim(options.cacheDir, s)
5770 return {"\\markdownRendererInputVerbatim{" ,name,""} }
5771 end
```

Define `writer->document` as a function that will transform a document `d` to the output format.

```
5772 function self.document(d)
5773 local buf = {"\\markdownRendererDocumentBegin\\n", d}
5774
5775 -- pop all attributes
5776 table.insert(buf, self.pop_attributes())
5777
5778 table.insert(buf, "\\markdownRendererDocumentEnd")
5779
5780 return buf
5781 end
```

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```
5782 local seen_identifiers = {}
5783 local key_value_regex = "([^\s=]+)%s*=%s*(.*)"
5784 local function normalize_attributes(attributes, auto_identifiers)
5785 -- normalize attributes
5786 local normalized_attributes = {}
5787 local has_explicit_identifiers = false
5788 local key, value
5789 for _, attribute in ipairs(attributes or {}) do
5790 if attribute:sub(1, 1) == "#" then
5791 table.insert(normalized_attributes, attribute)
5792 has_explicit_identifiers = true
5793 seen_identifiers[attribute:sub(2)] = true
5794 elseif attribute:sub(1, 1) == "." then
5795 table.insert(normalized_attributes, attribute)
5796 else
5797 key, value = attribute:match(key_value_regex)
5798 if key:lower() == "id" then
5799 table.insert(normalized_attributes, "#" .. value)
5800 elseif key:lower() == "class" then
5801 local classes = {}
5802 for class in value:gmatch("%S+") do
5803 table.insert(classes, class)
5804 end
5805 table.sort(classes)
5806 for _, class in ipairs(classes) do
5807 table.insert(normalized_attributes, "." .. class)
5808 end
5809 else
5810 table.insert(normalized_attributes, attribute)
5811 end
5812 end
5813 end
5814
5815 -- if no explicit identifiers exist, add auto identifiers
5816 if not has_explicit_identifiers and auto_identifiers ~= nil then
5817 local seen_auto_identifiers = {}
5818 for _, auto_identifier in ipairs(auto_identifiers) do
5819 if seen_auto_identifiers[auto_identifier] == nil then
5820 seen_auto_identifiers[auto_identifier] = true
5821 if seen_identifiers[auto_identifier] == nil then
5822 seen_identifiers[auto_identifier] = true
5823 table.insert(normalized_attributes,
5824 "#" .. auto_identifier)
5825 end
5826 end
5827 end
5828 local auto_identifier_number = 1
```

```

5827 while true do
5828 local numbered_auto_identifier = auto_identifier .. "-"
5829 .. auto_identifier_number
5830 if seen_identifiers[numbered_auto_identifier] == nil then
5831 seen_identifiers[numbered_auto_identifier] = true
5832 table.insert(normalized_attributes,
5833 "#" .. numbered_auto_identifier)
5834 break
5835 end
5836 auto_identifier_number = auto_identifier_number + 1
5837 end
5838 end
5839 end
5840 end
5841 end
5842
5843 -- sort and deduplicate normalized attributes
5844 table.sort(normalized_attributes)
5845 local seen_normalized_attributes = {}
5846 local deduplicated_normalized_attributes = {}
5847 for _, attribute in ipairs(normalized_attributes) do
5848 if seen_normalized_attributes[attribute] == nil then
5849 seen_normalized_attributes[attribute] = true
5850 table.insert(deduplicated_normalized_attributes, attribute)
5851 end
5852 end
5853
5854 return deduplicated_normalized_attributes
5855 end
5856
5857 function self.attributes(attributes, should_normalize_attributes)
5858 local normalized_attributes
5859 if should_normalize_attributes == false then
5860 normalized_attributes = attributes
5861 else
5862 normalized_attributes = normalize_attributes(attributes)
5863 end
5864
5865 local buf = {}
5866 local key, value
5867 for _, attribute in ipairs(normalized_attributes) do
5868 if attribute:sub(1, 1) == "#" then
5869 table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
5870 attribute:sub(2), "}"}))
5871 elseif attribute:sub(1, 1) == "." then
5872 table.insert(buf, {"\\markdownRendererAttributeName{",
5873 attribute:sub(2), "}"}))

```

```

5874 else
5875 key, value = attribute:match(key_value_regex)
5876 table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5877 key, "}{" , value, "}"}))
5878 end
5879 end
5880
5881 return buf
5882 end

```

Define `writer->active_attributes` as a stack of block-level attributes that are currently active. The `writer->active_attributes` member variable is mutable.

```

5883 self.active_attributes = {}

```

Define `writer->attribute_type_levels` as a hash table that maps attribute types to the number of attributes of said type in `writer->active_attributes`.

```

5884 self.attribute_type_levels = {}
5885 setmetatable(self.attribute_type_levels,
5886 { __index = function() return 0 end })

```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```

5887 local function apply_attributes()
5888 local buf = {}
5889 for i = 1, #self.active_attributes do
5890 local start_output = self.active_attributes[i][3]
5891 if start_output ~= nil then
5892 table.insert(buf, start_output)
5893 end
5894 end
5895 return buf
5896 end
5897
5898 local function tear_down_attributes()
5899 local buf = {}
5900 for i = #self.active_attributes, 1, -1 do
5901 local end_output = self.active_attributes[i][4]
5902 if end_output ~= nil then
5903 table.insert(buf, end_output)
5904 end
5905 end
5906 return buf
5907 end

```

The `writer->push_attributes` method adds `attributes` of type `attribute_type` to `writer->active_attributes`. The `start_output` string is used to construct a rope that will be returned by this function, together with output produced as a result

of slicing (see [slice](#)). The `end_output` string is stored together with `attributes` and is used to construct the return value of the `writer->pop_attributes` method.

```
5908 function self.push_attributes(attribute_type, attributes,
5909 start_output, end_output)
5910 local attribute_type_level = self.attribute_type_levels[attribute_type]
5911 self.attribute_type_levels[attribute_type] = attribute_type_level + 1
5912
5913 -- index attributes in a hash table for easy lookup
5914 attributes = attributes or {}
5915 for i = 1, #attributes do
5916 attributes[attributes[i]] = true
5917 end
5918
5919 local buf = {}
5920 -- handle slicing
5921 if attributes["#" .. self.slice_end_identifier] ~= nil and
5922 self.slice_end_type == "^" then
5923 if self.is_writing then
5924 table.insert(buf, self.undosep())
5925 table.insert(buf, tear_down_attributes())
5926 end
5927 self.is_writing = false
5928 end
5929 if attributes["#" .. self.slice_begin_identifier] ~= nil and
5930 self.slice_begin_type == "^" then
5931 table.insert(buf, apply_attributes())
5932 self.is_writing = true
5933 end
5934 if self.is_writing and start_output ~= nil then
5935 table.insert(buf, start_output)
5936 end
5937 table.insert(self.active_attributes,
5938 {attribute_type, attributes,
5939 start_output, end_output})
5940 return buf
5941 end
5942
```

The `writer->pop_attributes` method removes the most current of active block-level attributes from `writer->active_attributes` until attributes of type `attribute_type` have been removed. The method returns a rope constructed from the `end_output` string specified in the calls of `writer->push_attributes` that produced the most current attributes, and also from output produced as a result of slicing (see [slice](#)).

```
5943 function self.pop_attributes(attribute_type)
5944 local buf = {}
5945 -- pop attributes until we find attributes of correct type
```

```

5946 -- or until no attributes remain
5947 local current_attribute_type = false
5948 while current_attribute_type ~= attribute_type and
5949 #self.active_attributes > 0 do
5950 local attributes, _, end_output
5951 current_attribute_type, attributes, _, end_output = table.unpack(
5952 self.active_attributes[#self.active_attributes])
5953 local attribute_type_level = self.attribute_type_levels[current_attribute_type]
5954 self.attribute_type_levels[current_attribute_type] = attribute_type_level - 1
5955 if self.is_writing and end_output ~= nil then
5956 table.insert(buf, end_output)
5957 end
5958 table.remove(self.active_attributes, #self.active_attributes)
5959 -- handle slicing
5960 if attributes["#" .. self.slice_end_identifier] ~= nil
5961 and self.slice_end_type == "$" then
5962 if self.is_writing then
5963 table.insert(buf, self.undosep())
5964 table.insert(buf, tear_down_attributes())
5965 end
5966 self.is_writing = false
5967 end
5968 if attributes["#" .. self.slice_begin_identifier] ~= nil and
5969 self.slice_begin_type == "$" then
5970 self.is_writing = true
5971 table.insert(buf, apply_attributes())
5972 end
5973 end
5974 return buf
5975 end

```

Create an auto identifier string by stripping and converting characters from string `s`.

```

5976 local function create_auto_identifier(s)
5977 local buffer = {}
5978 local prev_space = false
5979 local letter_found = false
5980
5981 for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
5982 local char = utf8.char(code)
5983
5984 -- Remove everything up to the first letter.
5985 if not letter_found then
5986 local is_letter = unicode.utf8.match(char, "%a")
5987 if is_letter then
5988 letter_found = true
5989 else
5990 goto continue
5991 end

```

```

5992 end
5993
5994 -- Remove all non-alphanumeric characters, except underscores, hyphens, and per
5995 if not unicode.utf8.match(char, "[%w_-%.%s]") then
5996 goto continue
5997 end
5998
5999 -- Replace all spaces and newlines with hyphens.
6000 if unicode.utf8.match(char, "[%s\n]") then
6001 char = "-"
6002 if prev_space then
6003 goto continue
6004 else
6005 prev_space = true
6006 end
6007 else
6008 -- Convert all alphabetic characters to lowercase.
6009 char = unicode.utf8.lower(char)
6010 prev_space = false
6011 end
6012
6013 table.insert(buffer, char)
6014
6015 ::continue::
6016 end
6017
6018 if prev_space then
6019 table.remove(buffer)
6020 end
6021
6022 local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6023 return identifier
6024 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

6025 local function create_gfm_auto_identifier(s)
6026 local buffer = {}
6027 local prev_space = false
6028 local letter_found = false
6029
6030 for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
6031 local char = utf8.char(code)
6032
6033 -- Remove everything up to the first non-space.
6034 if not letter_found then
6035 local is_letter = unicode.utf8.match(char, "%S")

```



```

6036 if is_letter then
6037 letter_found = true
6038 else
6039 goto continue
6040 end
6041 end
6042
6043 -- Remove all non-alphanumeric characters, except underscores and hyphens.
6044 if not unicode.utf8.match(char, "[%w_-%s]") then
6045 prev_space = false
6046 goto continue
6047 end
6048
6049 -- Replace all spaces and newlines with hyphens.
6050 if unicode.utf8.match(char, "[%s\n]") then
6051 char = "-"
6052 if prev_space then
6053 goto continue
6054 else
6055 prev_space = true
6056 end
6057 else
6058 -- Convert all alphabetic characters to lowercase.
6059 char = unicode.utf8.lower(char)
6060 prev_space = false
6061 end
6062
6063 table.insert(buffer, char)
6064
6065 ::continue::
6066 end
6067
6068 if prev_space then
6069 table.remove(buffer)
6070 end
6071
6072 local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6073 return identifier
6074 end

```

Define `writer->heading` as a function that will transform an input heading `s` at level `level` with attributes `attributes` to the output format.

```

6075 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6076 self.secend_text = "\n\\markdownRendererSectionEnd "
6077 function self.heading(s, level, attributes)
6078 local buf = {}
6079 local flat_text, inlines = table.unpack(s)

```

```

6080
6081 -- push empty attributes for implied sections
6082 while self.attribute_type_levels["heading"] < level - 1 do
6083 table.insert(buf,
6084 self.push_attributes("heading",
6085 nil,
6086 self.secbegin_text,
6087 self.secend_text))
6088 end
6089
6090 -- pop attributes for sections that have ended
6091 while self.attribute_type_levels["heading"] >= level do
6092 table.insert(buf, self.pop_attributes("heading"))
6093 end
6094
6095 -- construct attributes for the new section
6096 local auto_identifiers = {}
6097 if self.options.autoIdentifiers then
6098 table.insert(auto_identifiers, create_auto_identifier(flat_text))
6099 end
6100 if self.options.gfmAutoIdentifiers then
6101 table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))
6102 end
6103 local normalized_attributes = normalize_attributes(attributes, auto_identifiers)
6104
6105 -- push attributes for the new section
6106 local start_output = {}
6107 local end_output = {}
6108 table.insert(start_output, self.secbegin_text)
6109 table.insert(end_output, self.secend_text)
6110
6111 table.insert(buf, self.push_attributes("heading",
6112 normalized_attributes,
6113 start_output,
6114 end_output))
6115 assert(self.attribute_type_levels["heading"] == level)
6116
6117 -- render the heading and its attributes
6118 if self.is_writing and #normalized_attributes > 0 then
6119 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\n")
6120 table.insert(buf, self.attributes(normalized_attributes, false))
6121 end
6122
6123 local cmd
6124 level = level + options.shiftHeadings
6125 if level <= 1 then
6126 cmd = "\\markdownRendererHeadingOne"

```

```

6127 elseif level == 2 then
6128 cmd = "\\markdownRendererHeadingTwo"
6129 elseif level == 3 then
6130 cmd = "\\markdownRendererHeadingThree"
6131 elseif level == 4 then
6132 cmd = "\\markdownRendererHeadingFour"
6133 elseif level == 5 then
6134 cmd = "\\markdownRendererHeadingFive"
6135 elseif level >= 6 then
6136 cmd = "\\markdownRendererHeadingSix"
6137 else
6138 cmd = ""
6139 end
6140 if self.is_writing then
6141 table.insert(buf, {cmd, "{", inlines, "}"}))
6142 end
6143
6144 if self.is_writing and #normalized_attributes > 0 then
6145 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd ")
6146 end
6147
6148 return buf
6149 end

```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```

6150 function self.get_state()
6151 return {
6152 is_writing=self.is_writing,
6153 flatten_inlines=self.flatten_inlines,
6154 active_attributes={table.unpack(self.active_attributes)},
6155 }
6156 end

```

Define `writer->set_state` as a function that restores the input state `s` and returns the previous state of the writer.

```

6157 function self.set_state(s)
6158 local previous_state = self.get_state()
6159 for key, value in pairs(s) do
6160 self[key] = value
6161 end
6162 return previous_state
6163 end

```

Define `writer->defer_call` as a function that will encapsulate the input function `f`, so that `f` is called with the state of the writer at the time of calling `writer->defer_call`.

```

6164 function self.defer_call(f)

```

```

6165 local previous_state = self.get_state()
6166 return function(...)
6167 local state = self.set_state(previous_state)
6168 local return_value = f(...)
6169 self.set_state(state)
6170 return return_value
6171 end
6172 end
6173
6174 return self
6175 end

```

### 3.1.4 Parsers

The `parsers` hash table stores PEG patterns that are static and can be reused between different `reader` objects.

```

6176 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

6177 parsers.percent = P("%")
6178 parsers.at = P("@")
6179 parsers.comma = P(",")
6180 parsers.asterisk = P("*")
6181 parsers.dash = P("-")
6182 parsers.plus = P("+")
6183 parsers.underscore = P("_")
6184 parsers.period = P(".")
6185 parsers.hash = P("#")
6186 parsers.dollar = P("$")
6187 parsers.ampersand = P("&")
6188 parsers.backtick = P("`")
6189 parsers.less = P("<")
6190 parsers.more = P(">")
6191 parsers.space = P(" ")
6192 parsers.squote = P("'")
6193 parsers.dquote = P('"')
6194 parsers.lparent = P("(")
6195 parsers.rparent = P(")")
6196 parsers.lbracket = P("[")
6197 parsers.rbracket = P("]")
6198 parsers.lbrace = P("{")
6199 parsers.rbrace = P("}")
6200 parsers.circumflex = P("^")
6201 parsers.slash = P("/")
6202 parsers.equal = P("=")
6203 parsers.colon = P(":")

```

```

6204 parsers.semicolon = P(";")
6205 parsers.exclamation = P("!")
6206 parsers.pipe = P("|")
6207 parsers.tilde = P("~")
6208 parsers.backslash = P("\\")
6209 parsers.tab = P("\t")
6210 parsers.newline = P("\n")
6211
6212 parsers.digit = R("09")
6213 parsers.hexdigit = R("09", "af", "AF")
6214 parsers.letter = R("AZ", "az")
6215 parsers.alphanumeric = R("AZ", "az", "09")
6216 parsers.keyword = parsers.letter
6217 * (parsers.alphanumeric + parsers.dash)^0
6218
6219 parsers.doubleasterisks = P("**")
6220 parsers.doubleunderscores = P("__")
6221 parsers.doubletildes = P("~~")
6222 parsers.fourspace = P(" ")
6223
6224 parsers.any = P(1)
6225 parsers.succeed = P(true)
6226 parsers.fail = P(false)
6227
6228 parsers.internal_punctuation = S(";,.?")
6229 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")

```

### 3.1.5 Unicode punctuation

This section documents the Unicode punctuation<sup>32</sup> recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

(CommonMark Spec, Version 0.31.2 (2024-01-28))

```

6230 parsers.punctuation = {}
6231 (function()
6232 local pathname = kpse.lookup("UnicodeData.txt")
6233 local file = assert(io.open(pathname, "r"),
6234 [[Could not open file "UnicodeData.txt"]])
6235 for line in file:lines() do
6236 local codepoint, major_category = line:match("^(%x+);[^\;]*;(%)a")
6237 if major_category == "P" or major_category == "S" then
6238 local code = unicode.utf8.char(tonumber(codepoint, 16))

```

<sup>32</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

6239 if parsers.punctuation[#code] == nil then
6240 parsers.punctuation[#code] = parsers.fail
6241 end
6242 local code_parser = parsers.succeed
6243 for i = 1, #code do
6244 local byte = code:sub(i, i)
6245 local byte_parser = S(byte)
6246 code_parser = code_parser
6247 * byte_parser
6248 end
6249 parsers.punctuation[#code] = parsers.punctuation[#code]
6250 + code_parser
6251 end
6252 end
6253 assert(file:close())
6254 end)()
6255
6256 parsers.escapable = parsers.ascii_punctuation
6257 parsers.anyescaped = parsers.backslash / "" * parsers.escapable
6258 + parsers.any
6259
6260 parsers.spacechar = S("\t ")
6261 parsers.spacing = S(" \n\r\t")
6262 parsers.nonspacechar = parsers.any - parsers.spacing
6263 parsers.optionalspace = parsers.spacechar^0
6264
6265 parsers.normalchar = parsers.any - (V("SpecialChar")
6266 + parsers.spacing)
6267 parsers.eof = -parsers.any
6268 parsers.nonindentspace = parsers.space^-3 * - parsers.spacechar
6269 parsers.indent = parsers.space^-3 * parsers.tab
6270 + parsers.fourspace / ""
6271 parsers.linechar = P(1 - parsers.newline)
6272
6273 parsers.blankline = parsers.optionalspace
6274 * parsers.newline / "\n"
6275 parsers.blanklines = parsers.blankline^0
6276 parsers.skipblanklines = (parsers.optionalspace * parsers.newline)^0
6277 parsers.indentedline = parsers.indent / ""
6278 * C(parsers.linechar^1 * parsers.newline^-
6279 1)
6279 parsers.optionallyindentedline = parsers.indent^-1 / ""
6280 * C(parsers.linechar^1 * parsers.newline^-
6281 1)
6281 parsers.sp = parsers.spacing^0
6282 parsers.spnl = parsers.optionalspace

```

```

6283 * (parsers.newline * parsers.optionalspace)^-
1
6284 parsers.line = parsers.linechar^0 * parsers.newline
6285 parsers.nonemptyline = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```

6286
6287 parsers.leader = parsers.space^-3
6288

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

6289 local function has_trail(indent_table)
6290 return indent_table ~= nil and
6291 indent_table.trail ~= nil and
6292 next(indent_table.trail) ~= nil
6293 end
6294

```

Check if indent table `indent_table` has any indents.

```

6295 local function has_indents(indent_table)
6296 return indent_table ~= nil and
6297 indent_table.indents ~= nil and
6298 next(indent_table.indents) ~= nil
6299 end
6300

```

Add a trail `trail_info` to the indent table `indent_table`.

```

6301 local function add_trail(indent_table, trail_info)
6302 indent_table.trail = trail_info
6303 return indent_table
6304 end
6305

```

Remove a trail `trail_info` from the indent table `indent_table`.

```

6306 local function remove_trail(indent_table)
6307 indent_table.trail = nil
6308 return indent_table
6309 end
6310

```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```

6311 local function update_indent_table(indent_table, new_indent, add)
6312 indent_table = remove_trail(indent_table)
6313
6314 if not has_indents(indent_table) then
6315 indent_table.indents = {}
6316 end
6317

```

```

6318
6319 if add then
6320 indent_table.indents[#indent_table.indents + 1] = new_indent
6321 else
6322 if indent_table.indents[#indent_table.indents].name == new_indent.name then
6323 indent_table.indents[#indent_table.indents] = nil
6324 end
6325 end
6326
6327 return indent_table
6328 end
6329

```

Remove an indent by its name `name`.

```

6330 local function remove_indent(name)
6331 local function remove_indent_level(s, i, indent_table) -- luacheck: ignore s i
6332 indent_table = update_indent_table(indent_table, {name=name}, false)
6333 return true, indent_table
6334 end
6335
6336 return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6337 end
6338

```

Process the spacing of a string of spaces and tabs `spacing` with preceding indent width from the start of the line `indent` and strip up to `left_strip_length` spaces. Return the remainder `remainder` and whether there is enough spaces to produce a code `is_code`. Return how many spaces were stripped, as well as if the minimum was met `is_minimum` and what remainder it left `minimum_remainder`.

```

6339 local function process_starter_spacing(indent, spacing, minimum, left_strip_length)
6340 left_strip_length = left_strip_length or 0
6341
6342 local count = 0
6343 local tab_value = 4 - (indent) % 4
6344
6345 local code_started, minimum_found = false, false
6346 local code_start, minimum_remainder = "", ""
6347
6348 local left_total_stripped = 0
6349 local full_remainder = ""
6350
6351 if spacing ~= nil then
6352 for i = 1, #spacing do
6353 local character = spacing:sub(i, i)
6354
6355 if character == "\t" then
6356 count = count + tab_value

```



```

6357 tab_value = 4
6358 elseif character == " " then
6359 count = count + 1
6360 tab_value = 4 - (1 - tab_value) % 4
6361 end
6362
6363 if (left_strip_length ~= 0) then
6364 local possible_to_strip = math.min(count, left_strip_length)
6365 count = count - possible_to_strip
6366 left_strip_length = left_strip_length - possible_to_strip
6367 left_total_stripped = left_total_stripped + possible_to_strip
6368 else
6369 full_remainder = full_remainder .. character
6370 end
6371
6372 if (minimum_found) then
6373 minimum_remainder = minimum_remainder .. character
6374 elseif (count >= minimum) then
6375 minimum_found = true
6376 minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6377 end
6378
6379 if (code_started) then
6380 code_start = code_start .. character
6381 elseif (count >= minimum + 4) then
6382 code_started = true
6383 code_start = code_start .. string.rep(" ", count - (minimum + 4))
6384 end
6385 end
6386 end
6387
6388 local remainder
6389 if (code_started) then
6390 remainder = code_start
6391 else
6392 remainder = string.rep(" ", count - minimum)
6393 end
6394
6395 local is_minimum = count >= minimum
6396 return {
6397 is_code = code_started,
6398 remainder = remainder,
6399 left_total_stripped = left_total_stripped,
6400 is_minimum = is_minimum,
6401 minimum_remainder = minimum_remainder,
6402 total_length = count,
6403 full_remainder = full_remainder

```

```

6404 }
6405 end
6406

```

Count the total width of all indents in the indent table `indent_table`.

```

6407 local function count_indent_tab_level(indent_table)
6408 local count = 0
6409 if not has_indents(indent_table) then
6410 return count
6411 end
6412
6413 for i=1, #indent_table.indents do
6414 count = count + indent_table.indents[i].length
6415 end
6416 return count
6417 end
6418

```

Count the total width of a delimiter `delimiter`.

```

6419 local function total_delimiter_length(delimiter)
6420 local count = 0
6421 if type(delimiter) == "string" then return #delimiter end
6422 for _, value in pairs(delimiter) do
6423 count = count + total_delimiter_length(value)
6424 end
6425 return count
6426 end
6427

```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```

6428 local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
6429 local last_trail = starter[1]
6430 local delimiter = starter[2]
6431 local raw_new_trail = starter[3]
6432
6433 if indent_type == "bq" and not breakable then
6434 indent_table.ignore_blockquote_blank = true
6435 end
6436
6437 if has_trail(indent_table) then
6438 local trail = indent_table.trail
6439 if trail.is_code then
6440 return false
6441 end
6442 last_trail = trail.remainder
6443 else
6444 local sp = process_starter_spacing(0, last_trail, 0, 0)

```

```

6445
6446 if sp.is_code then
6447 return false
6448 end
6449 last_trail = sp.remainder
6450 end
6451
6452 local preceding_indentation = count_indent_tab_level(indent_table) % 4
6453 local last_trail_length = #last_trail
6454 local delimiter_length = total_delimiter_length(delimiter)
6455
6456 local total_indent_level = preceding_indentation + last_trail_length + delimiter_length
6457
6458 local sp = {}
6459 if not is_blank then
6460 sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6461 end
6462
6463 local del_trail_length = sp.left_total_stripped
6464 if is_blank then
6465 del_trail_length = 1
6466 elseif not sp.is_code then
6467 del_trail_length = del_trail_length + #sp.remainder
6468 end
6469
6470 local indent_length = last_trail_length + delimiter_length + del_trail_length
6471 local new_indent_info = {name=indent_type, length=indent_length}
6472
6473 indent_table = update_indent_table(indent_table, new_indent_info, true)
6474 indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,
6475 full_remainder=sp.full_remainder})
6476
6477 return true, indent_table
6478 end
6479

```

Return the pattern corresponding with the indent name [name](#).

```

6480 local function decode_pattern(name)
6481 local delimiter = parsers.succeed
6482 if name == "bq" then
6483 delimiter = parsers.more
6484 end
6485
6486 return C(parsers.optionalspace) * C(delimiter) * C(parsers.optionalspace) * Cp()
6487 end
6488

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```
6489 local function left_blank_starter(indent_table)
6490 local blank_starter_index
6491
6492 if not has_indents(indent_table) then
6493 return
6494 end
6495
6496 for i = #indent_table.indents,1,-1 do
6497 local value = indent_table.indents[i]
6498 if value.name == "li" then
6499 blank_starter_index = i
6500 else
6501 break
6502 end
6503 end
6504
6505 return blank_starter_index
6506 end
6507
```

Apply the patterns decoded from the indents of the indent table `indent_table` iteratively starting at position `index` of the string `s`. If the `is_optional` mode is selected, match as many patterns as possible, else match all or fail. With the option `is_blank`, the parsing behaves as optional after the position of a blank-only indent has been surpassed.

```
6508 local function traverse_indent(s, i, indent_table, is_optional, is_blank, current_line_indents)
6509 local new_index = i
6510
6511 local preceding_indentation = 0
6512 local current_trail = {}
6513
6514 local blank_starter = left_blank_starter(indent_table)
6515
6516 if current_line_indents == nil then
6517 current_line_indents = {}
6518 end
6519
6520 for index = 1,#indent_table.indents do
6521 local value = indent_table.indents[index]
6522 local pattern = decode_pattern(value.name)
6523
6524 -- match decoded pattern
6525 local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6526 if new_indent_info == nil then
6527 local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
```

```

6528 if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end
6529 return is_optional, new_index, current_trail, current_line_indents
6530 end
6531
6532 return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optiona
6533 end
6534
6535 local raw_last_trail = new_indent_info[1]
6536 local delimiter = new_indent_info[2]
6537 local raw_new_trail = new_indent_info[3]
6538 local next_index = new_indent_info[4]
6539
6540 local space_only = delimiter == ""
6541
6542 -- check previous trail
6543 if not space_only and next(current_trail) == nil then
6544 local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6545 current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.to
6546 full_remainder=sp.full_remainder}
6547 end
6548
6549 if next(current_trail) ~= nil then
6550 if not space_only and current_trail.is_code then
6551 return is_optional, new_index, current_trail, current_line_indents
6552 end
6553 if current_trail.internal_remainder ~= nil then
6554 raw_last_trail = current_trail.internal_remainder
6555 end
6556 end
6557
6558 local raw_last_trail_length = 0
6559 local delimiter_length = 0
6560
6561 if not space_only then
6562 delimiter_length = #delimiter
6563 raw_last_trail_length = #raw_last_trail
6564 end
6565
6566 local total_indent_level = preceding_indentation + raw_last_trail_length + delimi
6567
6568 local spacing_to_process
6569 local minimum = 0
6570 local left_strip_length = 0
6571
6572 if not space_only then
6573 spacing_to_process = raw_new_trail
6574 left_strip_length = 1

```

```

6575 else
6576 spacing_to_process = raw_last_trail
6577 minimum = value.length
6578 end
6579
6580 local sp = process_starter_spacing(total_indent_level, spacing_to_process, minimum)
6581
6582 if space_only and not sp.is_minimum then
6583 return is_optional or (is_blank and blank_starter <= index), new_index, current_trail
6584 end
6585
6586 local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_spacing
6587
6588 -- update info for the next pattern
6589 if not space_only then
6590 preceding_indentation = preceding_indentation + indent_length
6591 else
6592 preceding_indentation = preceding_indentation + value.length
6593 end
6594
6595 current_trail = {is_code=sp.is_code, remainder=sp.remainder, internal_remainder=sp.internal_remainder,
6596 total_length=sp.total_length, full_remainder=sp.full_remainder}
6597
6598 current_line_indents[#current_line_indents + 1] = new_indent_info
6599 new_index = next_index
6600 end
6601
6602 return true, new_index, current_trail, current_line_indents
6603 end
6604

```

Check if a code trail is expected.

```

6605 local function check_trail(expect_code, is_code)
6606 return (expect_code and is_code) or (not expect_code and not is_code)
6607 end
6608

```

Check if the current trail of the `indent_table` would produce code if it is expected `expect_code` or it would not if it is not. If there is no trail, process and check the current spacing `spacing`.

```

6609 local function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_remainder)
6610 local is_code
6611 local remainder
6612
6613 if has_trail(indent_table) then
6614 local trail = indent_table.trail
6615 is_code = trail.is_code
6616 if is_code then

```

```

6617 remainder = trail.remainder
6618 else
6619 remainder = trail.full_remainder
6620 end
6621 else
6622 local sp = process_starter_spacing(0, spacing, 0, 0)
6623 is_code = sp.is_code
6624 if is_code then
6625 remainder = sp.remainder
6626 else
6627 remainder = sp.full_remainder
6628 end
6629 end
6630
6631 local result = check_trail(expect_code, is_code)
6632 if omit_remainder then
6633 return result
6634 end
6635 return result, remainder
6636 end
6637

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

6638 local function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:
6639 local trail
6640
6641 if has_trail(indent_table) then
6642 trail = indent_table.trail
6643 else
6644 trail = process_starter_spacing(0, spacing, 0, 0)
6645 end
6646
6647 local total_length = trail.total_length
6648 if total_length == nil then
6649 return false
6650 end
6651
6652 return min <= total_length and total_length <= max
6653 end
6654

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

6655 local function check_continuation_indentation(s, i, indent_table, is_optional, is_bla
6656 if not has_indents(indent_table) then
6657 return true
6658 end
6659

```

```

6660 local passes, new_index, current_trail, current_line_indents =
6661 traverse_indent(s, i, indent_table, is_optional, is_blank)
6662
6663 if passes then
6664 indent_table.current_line_indents = current_line_indents
6665 indent_table = add_trail(indent_table, current_trail)
6666 return new_index, indent_table
6667 end
6668 return false
6669 end
6670

```

Get name of the last indent from the `indent_table`.

```

6671 local function get_last_indent_name(indent_table)
6672 if has_indents(indent_table) then
6673 return indent_table.indents[#indent_table.indents].name
6674 end
6675 end
6676

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

6677 local function remove_remainder_if_blank(indent_table, remainder)
6678 if get_last_indent_name(indent_table) == "li" then
6679 return ""
6680 end
6681 return remainder
6682 end
6683

```

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```

6684 local function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore
6685 if trail == nil then
6686 trail = process_starter_spacing(0, spacing, 0, 0)
6687 end
6688
6689 if trail_type == "non-code" then
6690 return check_trail(false, trail.is_code)
6691 end
6692 if trail_type == "code" then
6693 return check_trail(true, trail.is_code)
6694 end
6695 if trail_type == "full-code" then
6696 if (trail.is_code) then
6697 return i, trail.remainder
6698 end
6699 return i, ""

```



```

6700 end
6701 if trail_type == "full-any" then
6702 return i, trail.internal_remainder
6703 end
6704 end
6705

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

6706 local function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
6707 if is_freezing then
6708 if indent_table.is_trail_frozen then
6709 indent_table.trail = indent_table.frozen_trail
6710 else
6711 indent_table.frozen_trail = indent_table.trail
6712 indent_table.is_trail_frozen = true
6713 end
6714 else
6715 indent_table.frozen_trail = nil
6716 indent_table.is_trail_frozen = false
6717 end
6718 return true, indent_table
6719 end
6720

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```

6721 local function check_continuation_indentation_and_trail(s, i, indent_table, is_optional,
6722 reset_rem, omit_remainder)
6723 if not has_indents(indent_table) then
6724 local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6725 local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, tra
6726 if remainder == nil then
6727 if result then
6728 return new_index
6729 end
6730 return false
6731 end
6732 if result then
6733 return new_index, remainder
6734 end
6735 return false
6736 end
6737
6738 local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt
6739
6740 if passes then
6741 local spacing

```

```

6742 if current_trail == nil then
6743 local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s,
6744 current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6745 new_index = newer_index
6746 spacing = newer_spacing
6747 else
6748 spacing = current_trail.remainder
6749 end
6750 local result, remainder = check_trail_type(s, new_index, current_trail, spacing,
6751 if remainder == nil or omit_remainder then
6752 if result then
6753 return new_index
6754 end
6755 return false
6756 end
6757
6758 if is_blank and reset_rem then
6759 remainder = remove_remainder_if_blank(indent_table, remainder)
6760 end
6761 if result then
6762 return new_index, remainder
6763 end
6764 return false
6765 end
6766 return false
6767 end
6768

```

The following patterns check whitespace indentation at the start of a block.

```

6769 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(false), che
6770
6771 parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals
6772
6773 parsers.check_code_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)
6774
6775 parsers.check_trail_length_range = function(min, max)
6776 return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
6777 end
6778
6779 parsers.check_trail_length = function(n)
6780 return parsers.check_trail_length_range(n, n)
6781 end
6782

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

6783 parsers.freeze_trail = Cg(Cmt(Cb("indent_info") * Cc(true), trail_freezing), "indent_
6784

```

```
6785 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde
6786
```

The following patterns check indentation in continuation lines as defined by the container start.

```
6787 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false), check_continuation_
6788
6789 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_
6790
6791 parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch
6792
```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```
6793
6794 parsers.check_minimal_indent_and_trail = Cmt(Cb("indent_info")
6795 * Cc(false) * Cc(false) * Cc("non-
code") * Cc(true),
6796 check_continuation_indentation_and_trail)
6797
6798 parsers.check_minimal_indent_and_code_trail = Cmt(Cb("indent_info")
6799 * Cc(false) * Cc(false) * Cc("code")
6800 check_continuation_indentation_and_t
6801
6802 parsers.check_minimal_blank_indent_and_full_code_trail = Cmt(Cb("indent_info")
6803 * Cc(false) * Cc(true) *
code") * Cc(true),
6804 check_continuation_indentation_and_full_code_trail)
6805
6806 parsers.check_minimal_indent_and_any_trail = Cmt(Cb("indent_info")
6807 * Cc(false) * Cc(false) * Cc("full-
any") * Cc(true) * Cc(false),
6808 check_continuation_indentation_and_any_trail)
6809
6810 parsers.check_minimal_blank_indent_and_any_trail = Cmt(Cb("indent_info")
6811 * Cc(false) * Cc(true) * Cc("full-
any") * Cc(true) * Cc(false),
6812 check_continuation_indentation_and_any_trail)
6813
6814 parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt(Cb("indent_info")
6815 * Cc(false) * Cc(true) * Cc("full-
any") * Cc(true) * Cc(true),
6816 check_continuation_indentation_and_any_trail_no_rem)
6817
6818 parsers.check_optional_indent_and_any_trail = Cmt(Cb("indent_info")
6819 * Cc(true) * Cc(false) * Cc("full-
any") * Cc(true) * Cc(false),
6820 check_continuation_indentation_and_any_trail)
```

```

6821
6822 parsers.check_optional_blank_indent_and_any_trail = Cmt(Cb("indent_info")
6823 * Cc(true) * Cc(true) * Cc("ful
any") * Cc(true) * Cc(false),
6824 check_continuation_indentation
6825

```

The following patterns specify behaviour around newlines.

```

6826
6827 parsers.spnlc_noexc = parsers.optionalspace
6828 * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
1
6829
6830 parsers.spnlc = parsers.optionalspace
6831 * (V("EndlineNoSub"))^-1
6832
6833 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
6834 + parsers.spacechar^1
6835
6836 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6837
6838 % \end{macrocode}
6839 % \begin{figure}
6840 % \hspace*{-0.1\textwidth}
6841 % \begin{minipage}{1.2\textwidth}
6842 % \centering
6843 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6844 % \node[state, initial by diamond, accepting] (noop) {initial};
6845 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
6846 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
6847 % \node[state] (comment) [below=of noop] {comment};
6848 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
6849 % \node[state] (blank_line) [below right=of comment] {blank line};
6850 % \path[->]
6851 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^
6852 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
6853 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
6854 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge$$\drsh
6855 % edge [bend left=10] node {match \drsh} (leading_spaces)
6856 % (leading_spaces) edge [loop below] node {match [\textvisiblespace\rightleftarrows
6857 % edge [bend right=90] node [right] {match \textbackslash} (odd_back
6858 % edge [bend left=10] node {match \%} (comment)
6859 % edge [bend right=10] node {ϵ} (blank_line)
6860 % edge [bend left=10] node [align=center, right=0.3cm] {match [$^\we
6861 % (blank_line) edge [loop below] node {match [\textvisiblespace\rightleftarrows]} (
6862 % edge [bend left=90] node [align=center, below=1.2cm] {match \drsh}
6863 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c

```

```

6864 % edge [bend right=10] node [align=center, above left=-
 0.3cm, xshift=0.1cm] {match [\$^\wedge$\textbackslash]\for \%, capture \textbackslash
6865 % (even_backslash) edge [bend left=10] node {\$\\epsilon$} (noop);
6866 % \end{tikzpicture}
6867 % \caption{A pushdown automaton that recognizes \TeX{} comments}
6868 % \label{fig:commented_line}
6869 % \end{minipage}
6870 % \end{figure}
6871 % \begin{markdown}
6872 %
6873 % The \luamdef{parsers.commented_line}^^1^ parser recognizes the regular
6874 % language of \TeX{} comments, see an equivalent finite automaton in Figure
6875 % <#fig:commented_line>.
6876 %
6877 % \end{markdown}
6878 % \begin{macrocode}
6879 parsers.commented_line_letter = parsers.linechar
6880 + parsers.newline
6881 - parsers.backslash
6882 - parsers.percent
6883 parsers.commented_line = Cg(Cc(""), "backslashes")
6884 * ((#(parsers.commented_line_letter
6885 - parsers.newline)
6886 * Cb("backslashes")
6887 * Cs(parsers.commented_line_letter
6888 - parsers.newline)^1 -- initial
6889 * Cg(Cc(""), "backslashes"))
6890 + #(parsers.backslash * parsers.backslash)
6891 * Cg((parsers.backslash -- even backslash
6892 * parsers.backslash)^1, "backslashes")
6893 + (parsers.backslash
6894 * (#parsers.percent
6895 * Cb("backslashes")
6896 / function(backslashes)
6897 return string.rep("\\", #backslashes / 2)
6898 end
6899 * C(parsers.percent)
6900 + #parsers.commented_line_letter
6901 * Cb("backslashes")
6902 * Cc("\\")
6903 * C(parsers.commented_line_letter))
6904 * Cg(Cc(""), "backslashes"))))^0
6905 * (#parsers.percent
6906 * Cb("backslashes")
6907 / function(backslashes)
6908 return string.rep("\\", #backslashes / 2)
6909 end

```

```

6910 * ((parsers.percent -- comment
6911 * parsers.line
6912 * #parsers.blankline) -- blank line
6913 / "\n"
6914 + parsers.percent -- comment
6915 * parsers.line
6916 * parsers.optionalspace) -- leading tabs and space
6917 + #(parsers.newline)
6918 * Cb("backslashes")
6919 * C(parsers.newline))
6920
6921 parsers.chunk = parsers.line * (parsers.optionallyindentedline
6922 - parsers.blankline)^0
6923
6924 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
6925 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
6926 parsers.attribute_key = (parsers.attribute_key_char
6927 - parsers.dash - parsers.digit)
6928 * parsers.attribute_key_char^0
6929 parsers.attribute_value = ((parsers.dquote / "\"")
6930 * (parsers.anyescaped - parsers.dquote)^0
6931 * (parsers.dquote / "\""))
6932 + ((parsers.squote / "\"")
6933 * (parsers.anyescaped - parsers.squote)^0
6934 * (parsers.squote / "\""))
6935 + (parsers.anyescaped - parsers.dquote - parsers.rbracket
6936 - parsers.space)^0
6937 parsers.attribute_identifier = parsers.attribute_key_char^1
6938 parsers.attribute_classname = parsers.letter
6939 * parsers.attribute_key_char^0
6940 parsers.attribute_raw = parsers.attribute_raw_char^1
6941
6942 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
6943 + C(parsers.hash
6944 * parsers.attribute_identifier)
6945 + C(parsers.period
6946 * parsers.attribute_classname)
6947 + Cs(parsers.attribute_key
6948 * parsers.optionalspace * parsers.equal * parsers.optionalspace
6949 * parsers.attribute_value)
6950 parsers.attributes = parsers.lbrace
6951 * parsers.optionalspace
6952 * parsers.attribute
6953 * (parsers.spacechar^1
6954 * parsers.attribute)^0
6955 * parsers.optionalspace
6956 * parsers.rbrace

```

```

6957
6958
6959 parsers.raw_attribute = parsers.lbrace
6960 * parsers.optionalspace
6961 * parsers.equal
6962 * C(parsers.attribute_raw)
6963 * parsers.optionalspace
6964 * parsers.rbrace
6965
6966 -- block followed by 0 or more optionally
6967 -- indented blocks with first line indented.
6968 parsers.indented_blocks = function(bl)
6969 return Cs(bl
6970 * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
6971 * (parsers.blankline^1 + parsers.eof))
6972 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

6973 local function repeat_between(pattern, min, max)
6974 return -pattern^(max + 1) * pattern^min
6975 end
6976
6977 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
6978 * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
6979 parsers.decentity = parsers.ampersand * parsers.hash
6980 * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
6981 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
6982 * parsers.semicolon
6983
6984 parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
6985 + parsers.decentity / entities.dec_entity
6986 + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

6987 parsers.bullet = function(bullet_char, interrupting)
6988 local allowed_end
6989 if interrupting then
6990 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
6991 else
6992 allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
6993 end
6994 return parsers.check_trail
6995 * Ct(C(bullet_char) * Cc(""))
6996 * allowed_end
6997 end
6998

```

```

6999 local function tickbox(interior)
7000 return parsers.optionalspace * parsers.lbracket
7001 * interior * parsers.rbracket * parsers.spacechar^1
7002 end
7003
7004 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7005 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7006 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7007

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```

7008 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7009
7010 local function captures_equal_length(_,i,a,b)
7011 return #a == #b and i
7012 end
7013
7014 parsers.closeticks = Cmt(C(parsers.backtick^1)
7015 * Cb("ticks"), captures_equal_length)
7016
7017 parsers.intickschar = (parsers.any - S("\n\r`"))
7018 + V("NoSoftLineBreakEndline")
7019 + (parsers.backtick^1 - parsers.closeticks)
7020
7021 local function process_inticks(s)
7022 s = s:gsub("\n", " ")
7023 s = s:gsub("^ (.*) $", "%1")
7024 return s
7025 end
7026
7027 parsers.inticks = parsers.openticks
7028 * C(parsers.space^0)
7029 * parsers.closeticks
7030 + parsers.openticks
7031 * Cs(Cs(parsers.intickschar^0) / process_inticks)
7032 * parsers.closeticks
7033

```

### 3.1.5.5 Parsers Used for HTML

```

7034 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
7035 parsers.keyword_exact = function(s)
7036 local parser = P(0)
7037 for i=1,#s do
7038 local c = s:sub(i,i)
7039 local m = c .. upper(c)
7040 parser = parser * S(m)

```



```
7041 end
7042 return parser
7043 end
7044
7045 parsers.special_block_keyword =
7046 parsers.keyword_exact("pre") +
7047 parsers.keyword_exact("script") +
7048 parsers.keyword_exact("style") +
7049 parsers.keyword_exact("textarea")
7050
7051 parsers.block_keyword =
7052 parsers.keyword_exact("address") +
7053 parsers.keyword_exact("article") +
7054 parsers.keyword_exact("aside") +
7055 parsers.keyword_exact("base") +
7056 parsers.keyword_exact("basefont") +
7057 parsers.keyword_exact("blockquote") +
7058 parsers.keyword_exact("body") +
7059 parsers.keyword_exact("caption") +
7060 parsers.keyword_exact("center") +
7061 parsers.keyword_exact("col") +
7062 parsers.keyword_exact("colgroup") +
7063 parsers.keyword_exact("dd") +
7064 parsers.keyword_exact("details") +
7065 parsers.keyword_exact("dialog") +
7066 parsers.keyword_exact("dir") +
7067 parsers.keyword_exact("div") +
7068 parsers.keyword_exact("dl") +
7069 parsers.keyword_exact("dt") +
7070 parsers.keyword_exact("fieldset") +
7071 parsers.keyword_exact("figcaption") +
7072 parsers.keyword_exact("figure") +
7073 parsers.keyword_exact("footer") +
7074 parsers.keyword_exact("form") +
7075 parsers.keyword_exact("frame") +
7076 parsers.keyword_exact("frameset") +
7077 parsers.keyword_exact("h1") +
7078 parsers.keyword_exact("h2") +
7079 parsers.keyword_exact("h3") +
7080 parsers.keyword_exact("h4") +
7081 parsers.keyword_exact("h5") +
7082 parsers.keyword_exact("h6") +
7083 parsers.keyword_exact("head") +
7084 parsers.keyword_exact("header") +
7085 parsers.keyword_exact("hr") +
7086 parsers.keyword_exact("html") +
7087 parsers.keyword_exact("iframe") +
```

```

7088 parsers.keyword_exact("legend") +
7089 parsers.keyword_exact("li") +
7090 parsers.keyword_exact("link") +
7091 parsers.keyword_exact("main") +
7092 parsers.keyword_exact("menu") +
7093 parsers.keyword_exact("menuitem") +
7094 parsers.keyword_exact("nav") +
7095 parsers.keyword_exact("noframes") +
7096 parsers.keyword_exact("ol") +
7097 parsers.keyword_exact("optgroup") +
7098 parsers.keyword_exact("option") +
7099 parsers.keyword_exact("p") +
7100 parsers.keyword_exact("param") +
7101 parsers.keyword_exact("section") +
7102 parsers.keyword_exact("source") +
7103 parsers.keyword_exact("summary") +
7104 parsers.keyword_exact("table") +
7105 parsers.keyword_exact("tbody") +
7106 parsers.keyword_exact("td") +
7107 parsers.keyword_exact("tfoot") +
7108 parsers.keyword_exact("th") +
7109 parsers.keyword_exact("thead") +
7110 parsers.keyword_exact("title") +
7111 parsers.keyword_exact("tr") +
7112 parsers.keyword_exact("track") +
7113 parsers.keyword_exact("ul")
7114
7115 -- end conditions
7116 parsers.html_blankline_end_condition = parsers.linechar^0
7117 * (parsers.newline
7118 * (parsers.check_minimal_blank_indent_and_any
7119 * #parsers.blankline
7120 + parsers.check_minimal_indent_and_any_trai
7121 * parsers.linechar^1)^0
7122 * (parsers.newline^-1 / "")
7123
7124 local function remove_trailing_blank_lines(s)
7125 return s:gsub("[\n\r]+%s*$", "")
7126 end
7127
7128 parsers.html_until_end = function(end_marker)
7129 return Cs(Cs((parsers.newline
7130 * (parsers.check_minimal_blank_indent_and_any_trail
7131 * #parsers.blankline
7132 + parsers.check_minimal_indent_and_any_trail)
7133 + parsers.linechar - end_marker)^0
7134 * parsers.linechar^0 * parsers.newline^-1)

```

```

7135 / remove_trailing_blank_lines)
7136 end
7137
7138 -- attributes
7139 parsers.html_attribute_spacing = parsers.optionalspace
7140 * V("NoSoftLineBreakEndline")
7141 * parsers.optionalspace
7142 + parsers.spacechar^1
7143
7144 parsers.html_attribute_name = (parsers.letter + parsers.colon + parsers.underscore)
7145 * (parsers.alphanumeric + parsers.colon + parsers.undersco
7146 + parsers.period + parsers.dash)^0
7147
7148 parsers.html_attribute_value = parsers.squote
7149 * (parsers.linechar - parsers.squote)^0
7150 * parsers.squote
7151 + parsers.dquote
7152 * (parsers.linechar - parsers.dquote)^0
7153 * parsers.dquote
7154 + (parsers.any - parsers.spacechar - parsers.newline
7155 - parsers.dquote - parsers.squote - parsers.backtick
7156 - parsers.equal - parsers.less - parsers.more)^1
7157
7158 parsers.html_inline_attribute_value = parsers.squote
7159 * (V("NoSoftLineBreakEndline")
7160 + parsers.any
7161 - parsers.blankline^2
7162 - parsers.squote)^0
7163 * parsers.squote
7164 + parsers.dquote
7165 * (V("NoSoftLineBreakEndline")
7166 + parsers.any
7167 - parsers.blankline^2
7168 - parsers.dquote)^0
7169 * parsers.dquote
7170 + (parsers.any - parsers.spacechar - parsers.newl
7171 - parsers.dquote - parsers.squote - parsers.bac
7172 - parsers.equal - parsers.less - parsers.more)^1
7173
7174 parsers.html_attribute_value_specification = parsers.optionalspace
7175 * parsers.equal
7176 * parsers.optionalspace
7177 * parsers.html_attribute_value
7178
7179 parsers.html_spnl = parsers.optionalspace
7180 * (V("NoSoftLineBreakEndline") * parsers.optionalspace)^-
1

```

```

7181
7182 parsers.html_inline_attribute_value_specification = parsers.html_spnl
7183 * parsers.equal
7184 * parsers.html_spnl
7185 * parsers.html_inline_attribute_val
7186
7187 parsers.html_attribute = parsers.html_attribute_spacing
7188 * parsers.html_attribute_name
7189 * parsers.html_inline_attribute_value_specification^-
7190
7191 parsers.html_non_newline_attribute = parsers.spacechar^1
7192 * parsers.html_attribute_name
7193 * parsers.html_attribute_value_specification^-
7194
7195 parsers.nested_breaking_blank = parsers.newline
7196 * parsers.check_minimal_blank_indent
7197 * parsers.blankline
7198
7199 parsers.html_comment_start = P("<!--")
7200
7201 parsers.html_comment_end = P("-->")
7202
7203 parsers.html_comment = Cs(parsers.html_comment_start
7204 * parsers.html_until_end(parsers.html_comment_end))
7205
7206 parsers.html_inline_comment = (parsers.html_comment_start / "")
7207 * -P(">") * -P("->")
7208 * Cs((V("NoSoftLineBreakEndline") + parsers.any
7209 - parsers.nested_breaking_blank - parsers.html_commen
7210 * (parsers.html_comment_end / ""))
7211
7212 parsers.html_cdatasection_start = P("<![CDATA[")
7213
7214 parsers.html_cdatasection_end = P("]]>")
7215
7216 parsers.html_cdatasection = Cs(parsers.html_cdatasection_start
7217 * parsers.html_until_end(parsers.html_cdatasection_end))
7218
7219 parsers.html_inline_cdatasection = parsers.html_cdatasection_start
7220 * Cs((V("NoSoftLineBreakEndline") + parsers.any
7221 - parsers.nested_breaking_blank - parsers.html_
7222 * parsers.html_cdatasection_end)
7223
7224 parsers.html_declaration_start = P("<!") * parsers.letter
7225

```

```

7226 parsers.html_declaration_end = P(">")
7227
7228 parsers.html_declaration = Cs(parsers.html_declaration_start
7229 * parsers.html_until_end(parsers.html_declaration_end))
7230
7231 parsers.html_inline_declaration = parsers.html_declaration_start
7232 * Cs(V("NoSoftLineBreakEndline") + parsers.any
7233 - parsers.nested_breaking_blank - parsers.html_de
7234 * parsers.html_declaration_end
7235
7236 parsers.html_instruction_start = P("<?")
7237
7238 parsers.html_instruction_end = P("?>")
7239
7240 parsers.html_instruction = Cs(parsers.html_instruction_start
7241 * parsers.html_until_end(parsers.html_instruction_end))
7242
7243 parsers.html_inline_instruction = parsers.html_instruction_start
7244 * Cs(V("NoSoftLineBreakEndline") + parsers.any
7245 - parsers.nested_breaking_blank - parsers.html_in
7246 * parsers.html_instruction_end
7247
7248 parsers.html_blankline = parsers.newline
7249 * parsers.optionalspace
7250 * parsers.newline
7251
7252 parsers.html_tag_start = parsers.less
7253
7254 parsers.html_tag_closing_start = parsers.less
7255 * parsers.slash
7256
7257 parsers.html_tag_end = parsers.html_spnl
7258 * parsers.more
7259
7260 parsers.html_empty_tag_end = parsers.html_spnl
7261 * parsers.slash
7262 * parsers.more
7263
7264 -- opening tags
7265 parsers.html_any_open_inline_tag = parsers.html_tag_start
7266 * parsers.keyword
7267 * parsers.html_attribute^0
7268 * parsers.html_tag_end
7269
7270 parsers.html_any_open_tag = parsers.html_tag_start
7271 * parsers.keyword
7272 * parsers.html_non_newline_attribute^0

```

```

7273 * parsers.html_tag_end
7274
7275 parsers.html_open_tag = parsers.html_tag_start
7276 * parsers.block_keyword
7277 * parsers.html_attribute^0
7278 * parsers.html_tag_end
7279
7280 parsers.html_open_special_tag = parsers.html_tag_start
7281 * parsers.special_block_keyword
7282 * parsers.html_attribute^0
7283 * parsers.html_tag_end
7284
7285 -- incomplete tags
7286 parsers.incomplete_tag_following = parsers.spacechar
7287 + parsers.more
7288 + parsers.slash * parsers.more
7289 + #(parsers.newline + parsers.eof)
7290
7291 parsers.incomplete_special_tag_following = parsers.spacechar
7292 + parsers.more
7293 + #(parsers.newline + parsers.eof)
7294
7295 parsers.html_incomplete_open_tag = parsers.html_tag_start
7296 * parsers.block_keyword
7297 * parsers.incomplete_tag_following
7298
7299 parsers.html_incomplete_open_special_tag = parsers.html_tag_start
7300 * parsers.special_block_keyword
7301 * parsers.incomplete_special_tag_following
7302
7303 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7304 * parsers.block_keyword
7305 * parsers.incomplete_tag_following
7306
7307 parsers.html_incomplete_close_special_tag = parsers.html_tag_closing_start
7308 * parsers.special_block_keyword
7309 * parsers.incomplete_tag_following
7310
7311 -- closing tags
7312 parsers.html_close_tag = parsers.html_tag_closing_start
7313 * parsers.block_keyword
7314 * parsers.html_tag_end
7315
7316 parsers.html_any_close_tag = parsers.html_tag_closing_start
7317 * parsers.keyword
7318 * parsers.html_tag_end
7319

```

```

7320 parsers.html_close_special_tag = parsers.html_tag_closing_start
7321 * parsers.special_block_keyword
7322 * parsers.html_tag_end
7323
7324 -- empty tags
7325 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7326 * parsers.keyword
7327 * parsers.html_attribute^0
7328 * parsers.html_empty_tag_end
7329
7330 parsers.html_any_empty_tag = parsers.html_tag_start
7331 * parsers.keyword
7332 * parsers.html_non_newline_attribute^0
7333 * parsers.optionalspace
7334 * parsers.slash
7335 * parsers.more
7336
7337 parsers.html_empty_tag = parsers.html_tag_start
7338 * parsers.block_keyword
7339 * parsers.html_attribute^0
7340 * parsers.html_empty_tag_end
7341
7342 parsers.html_empty_special_tag = parsers.html_tag_start
7343 * parsers.special_block_keyword
7344 * parsers.html_attribute^0
7345 * parsers.html_empty_tag_end
7346
7347 parsers.html_incomplete_blocks = parsers.html_incomplete_open_tag
7348 + parsers.html_incomplete_open_special_tag
7349 + parsers.html_incomplete_close_tag
7350
7351 -- parse special html blocks
7352 parsers.html_blankline_ending_special_block_opening = (parsers.html_close_special_tag
7353 + parsers.html_empty_special_tag
7354 * #(parsers.optionalspace
7355 * (parsers.newline + parsers.e
7356
7357 parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_block_opening
7358 * parsers.html_blankline_end_condition
7359
7360 parsers.html_special_block_opening = parsers.html_incomplete_open_special_tag
7361 - parsers.html_empty_special_tag
7362
7363 parsers.html_closing_special_block = parsers.html_special_block_opening
7364 * parsers.html_until_end(parsers.html_close_special_block)
7365
7366 parsers.html_special_block = parsers.html_blankline_ending_special_block

```

```

7367 + parsers.html_closing_special_block
7368
7369 -- parse html blocks
7370 parsers.html_block_opening = parsers.html_incomplete_open_tag
7371 + parsers.html_incomplete_close_tag
7372
7373 parsers.html_block = parsers.html_block_opening
7374 * parsers.html_blankline_end_condition
7375
7376 -- parse any html blocks
7377 parsers.html_any_block_opening = (parsers.html_any_open_tag
7378 + parsers.html_any_close_tag
7379 + parsers.html_any_empty_tag)
7380 * #(parsers.optionalspace * (parsers.newline + parser
7381
7382 parsers.html_any_block = parsers.html_any_block_opening
7383 * parsers.html_blankline_end_condition
7384
7385 parsers.html_inline_comment_full = parsers.html_comment_start
7386 * -P(">") * -P("->")
7387 * Cs((V("NoSoftLineBreakEndline") + parsers.any - P
7388 ")
7389 - parsers.nested_breaking_blank - parsers.html_
7390 * parsers.html_comment_end
7391
7392 parsers.html_inline_tags = parsers.html_inline_comment_full
7393 + parsers.html_any_empty_inline_tag
7394 + parsers.html_inline_instruction
7395 + parsers.html_inline_cdatasection
7396 + parsers.html_inline_declaration
7397 + parsers.html_any_open_inline_tag
7398 + parsers.html_any_close_tag

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

7399 parsers.urlchar = parsers.anyescaped
7400 - parsers.newline
7401 - parsers.more
7402
7403 parsers.auto_link_scheme_part = parsers.alphanumeric
7404 + parsers.plus
7405 + parsers.period
7406 + parsers.dash
7407
7408 parsers.auto_link_scheme = parsers.letter
7409 * parsers.auto_link_scheme_part

```



```

7410 * parsers.auto_link_scheme_part^-30
7411
7412 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7413 * (parsers.any - parsers.spacing - parsers.less - parsers.more)
7414
7415 parsers.printable_characters = S("!.#%&'*/=?^_`{|}~-")
7416
7417 parsers.email_address_local_part_char = parsers.alphanumeric
7418 + parsers.printable_characters
7419
7420 parsers.email_address_local_part = parsers.email_address_local_part_char^1
7421
7422 parsers.email_address_dns_label = parsers.alphanumeric
7423 * (parsers.alphanumeric + parsers.dash)^-
7424 62
7425 * B(parsers.alphanumeric)
7426
7426 parsers.email_address_domain = parsers.email_address_dns_label
7427 * (parsers.period * parsers.email_address_dns_label)^0
7428
7429 parsers.email_address = parsers.email_address_local_part
7430 * parsers.at
7431 * parsers.email_address_domain
7432
7433 parsers.auto_link_url = parsers.less
7434 * C(parsers.absolute_uri)
7435 * parsers.more
7436
7437 parsers.auto_link_email = parsers.less
7438 * C(parsers.email_address)
7439 * parsers.more
7440
7441 parsers.auto_link_relative_reference = parsers.less
7442 * C(parsers.urlchar^1)
7443 * parsers.more
7444
7445 parsers.autolink = parsers.auto_link_url
7446 + parsers.auto_link_email
7447
7448 -- content in balanced brackets, parentheses, or quotes:
7449 parsers.bracketed = P{ parsers.lbracket
7450 * ((parsers.backslash / "" * parsers.rbracket
7451 + parsers.any - (parsers.lbracket
7452 + parsers.rbracket
7453 + parsers.blankline^2)
7454) + V(1))^0
7455 * parsers.rbracket }

```

```

7456
7457 parsers.inparens = P{ parsers.lparent
7458 * ((parsers.anyescaped - (parsers.lparent
7459 + parsers.rparent
7460 + parsers.blankline^2)
7461) + V(1))^0
7462 * parsers.rparent }
7463
7464 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
7465 * ((parsers.anyescaped - (parsers.squote
7466 + parsers.blankline^2)
7467) + V(1))^0
7468 * parsers.squote }
7469
7470 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
7471 * ((parsers.anyescaped - (parsers.dquote
7472 + parsers.blankline^2)
7473) + V(1))^0
7474 * parsers.dquote }
7475
7476 parsers.link_text = parsers.lbracket
7477 * Cs((parsers.alphanumeric^1
7478 + parsers.bracketed
7479 + parsers.inticks
7480 + parsers.autolink
7481 + V("InlineHtml")
7482 + (parsers.backslash * parsers.backslash)
7483 + (parsers.backslash * (parsers.lbracket + parsers.rbracket)
7484 + V("NoSoftLineBreakSpace")
7485 + V("NoSoftLineBreakEndline")
7486 + (parsers.any
7487 - (parsers.newline + parsers.lbracket + parsers.rbracket)
7488 * parsers.rbracket
7489
7490 parsers.link_label = parsers.lbracket
7491 * -(parsers.sp * parsers.rbracket)
7492 * #((parsers.any - parsers.rbracket)^-999 * parsers.rbracket)
7493 * Cs((parsers.alphanumeric^1
7494 + parsers.inticks
7495 + parsers.autolink
7496 + V("InlineHtml")
7497 + (parsers.backslash * parsers.backslash)
7498 + (parsers.backslash * (parsers.lbracket + parsers.rbracket)
7499 + V("NoSoftLineBreakSpace")
7500 + V("NoSoftLineBreakEndline")
7501 + (parsers.any
7502 - (parsers.newline + parsers.lbracket + parsers.rbracket)

```

```

7503 * parsers.rbracket
7504
7505 parsers.inparens_url = P{ parsers.lparent
7506 * ((parsers.anyescaped - (parsers.lparent
7507 + parsers.rparent
7508 + parsers.spacing)
7509) + V(1))^0
7510 * parsers.rparent }
7511
7512 -- url for markdown links, allowing nested brackets:
7513 parsers.url = parsers.less * Cs((parsers.anyescaped
7514 - parsers.newline
7515 - parsers.less
7516 - parsers.more)^0)
7517 * parsers.more
7518 + -parsers.less
7519 * Cs((parsers.inparens_url + (parsers.anyescaped
7520 - parsers.spacing
7521 - parsers.lparent
7522 - parsers.rparent))^1)
7523
7524 -- quoted text:
7525 parsers.title_s = parsers.squote
7526 * Cs((parsers.html_entities
7527 + V("NoSoftLineBreakSpace")
7528 + V("NoSoftLineBreakEndline")
7529 + (parsers.anyescaped - parsers.newline - parsers.squote - p
7530 * parsers.squote
7531
7532 parsers.title_d = parsers.dquote
7533 * Cs((parsers.html_entities
7534 + V("NoSoftLineBreakSpace")
7535 + V("NoSoftLineBreakEndline")
7536 + (parsers.anyescaped - parsers.newline - parsers.dquote - p
7537 * parsers.dquote
7538
7539 parsers.title_p = parsers.lparent
7540 * Cs((parsers.html_entities
7541 + V("NoSoftLineBreakSpace")
7542 + V("NoSoftLineBreakEndline")
7543 + (parsers.anyescaped - parsers.newline - parsers.lparent -
7544 - parsers.blankline^2))^0)
7545 * parsers.rparent
7546
7547 parsers.title = parsers.title_d + parsers.title_s + parsers.title_p
7548
7549 parsers.optionaltitle

```

```

7550 = parsers.spnlc * parsers.title * parsers.spacechar^0
7551 + Cc("")
7552

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

7553 -- parse a reference definition: [foo]: /bar "title"
7554 parsers.define_reference_parser = (parsers.check_trail / "") * parsers.link_label * p
7555 * parsers.spnlc * parsers.url
7556 * (parsers.spnlc_sep * parsers.title * parsers.only_
7557 + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

7558 parsers.Inline = V("Inline")
7559
7560 -- parse many p between starter and ender
7561 parsers.between = function(p, starter, ender)
7562 local ender2 = B(parsers.nospacechar) * ender
7563 return (starter * #parsers.nospacechar * Ct(p * (p - ender2)^0) * ender2)
7564 end
7565

```

### 3.1.5.9 Block Elements

```

7566 parsers.lineof = function(c)
7567 return (parsers.check_trail_no_rem * (P(c) * parsers.optionalspace)^3
7568 * (parsers.newline + parsers.eof))
7569 end
7570
7571 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
7572 + parsers.lineof(parsers.dash)
7573 + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

7574 -- parse Atx heading start and return level
7575 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
7576 * -parsers.hash / length
7577
7578 -- parse setext header ending and return level
7579 parsers.heading_level = parsers.nonindentospace * parsers.equal^1 * parsers.optionalsp
7580 + parsers.nonindentospace * parsers.dash^1 * parsers.optionalspa
7581
7582 local function strip_atx_end(s)
7583 return s:gsub("%s+#+%s*\n$", "")
7584 end
7585
7586 parsers.atx_heading = parsers.check_trail_no_rem

```

```

7587 * Cg(parsers.heading_start, "level")
7588 * (C(parsers.optionalspace
7589 * parsers.hash^0
7590 * parsers.optionalspace
7591 * parsers.newline)
7592 + parsers.spacechar^1
7593 * C(parsers.line))

```

### 3.1.6 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new TeX reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```

7594 M.reader = {}
7595 function M.reader.new(writer, options)
7596 local self = {}

```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```

7597 self.writer = writer
7598 self.options = options

```

Create a `reader->parsers` hash table that stores PEG patterns that depend on the received `options`. Make `reader->parsers` inherit from the global `parsers` table.

```

7599 self.parsers = {}
7600 (function(parsers)
7601 setmetatable(self.parsers, {
7602 __index = function (_, key)
7603 return parsers[key]
7604 end
7605 })
7606 end)(parsers)

```

Make `reader->parsers` available as a local `parsers` variable that will shadow the global `parsers` table and will make `reader->parsers` easier to type in the rest of the reader code.

```

7607 local parsers = self.parsers

```

### 3.1.6.1 Top-Level Helper Functions

Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
7608 function self.normalize_tag(tag)
7609 tag = util.ropetostring(tag)
7610 tag = tag:gsub("[\n\r\t]+", " ")
7611 tag = tag:gsub("^ ", ""):gsub(" $", "")
7612 tag = uni_algos.case.casefold(tag, true, false)
7613 return tag
7614 end
```

Define `iterlines` as a function that iterates over the lines of the input string `s`, transforms them using an input function `f`, and reassembles them into a new string, which it returns.

```
7615 local function iterlines(s, f)
7616 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7617 return util.ropetostring(rope)
7618 end
```

Define `expandtabs` either as an identity function, when the `preserveTabs` Lua interface option is enabled, or to a function that expands tabs into spaces otherwise.

```
7619 if options.preserveTabs then
7620 self.expandtabs = function(s) return s end
7621 else
7622 self.expandtabs = function(s)
7623 if s:find("\t") then
7624 return iterlines(s, util.expand_tabs_in_line)
7625 else
7626 return s
7627 end
7628 end
7629 end
```

### 3.1.6.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `oplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
7630 self.parser_functions = {}
7631 self.create_parser = function(name, grammar, toplevel)
7632 self.parser_functions[name] = function(str)
```

If the parser function is top-level and the `stripIndent` Lua option is enabled, we will first expand tabs in the input string `str` into spaces and then we will count

the minimum indent across all lines, skipping blank lines. Next, we will remove the minimum indent from all lines.

```

7633 if toplevel and options.stripIndent then
7634 local min_prefix_length, min_prefix = nil, ''
7635 str = iterlines(str, function(line)
7636 if lpeg.match(parsers.nonemptyline, line) == nil then
7637 return line
7638 end
7639 line = util.expand_tabs_in_line(line)
7640 local prefix = lpeg.match(C(parsers.optionalspace), line)
7641 local prefix_length = #prefix
7642 local is_shorter = min_prefix_length == nil
7643 is_shorter = is_shorter or prefix_length < min_prefix_length
7644 if is_shorter then
7645 min_prefix_length, min_prefix = prefix_length, prefix
7646 end
7647 return line
7648 end)
7649 str = str:gsub('^' .. min_prefix, '')
7650 end

```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain T<sub>E</sub>X comments from the input string `str` together with the trailing newline characters.

```

7651 if toplevel and (options.texComments or options.hybrid) then
7652 str = lpeg.match(Ct(parsers.commented_line^1), str)
7653 str = util.rope_to_string(str)
7654 end
7655 local res = lpeg.match(grammar(), str)
7656 if res == nil then
7657 error(format("%s failed on:\n%s", name, str:sub(1,20)))
7658 else
7659 return res
7660 end
7661 end
7662 end
7663
7664 self.create_parser("parse_blocks",
7665 function()
7666 return parsers.blocks
7667 end, true)
7668
7669 self.create_parser("parse_blocks_nested",
7670 function()
7671 return parsers.blocks_nested
7672 end, false)
7673

```

```

7674 self.create_parser("parse_inlines",
7675 function()
7676 return parsers.inlines
7677 end, false)
7678
7679 self.create_parser("parse_inlines_no_inline_note",
7680 function()
7681 return parsers.inlines_no_inline_note
7682 end, false)
7683
7684 self.create_parser("parse_inlines_no_html",
7685 function()
7686 return parsers.inlines_no_html
7687 end, false)
7688
7689 self.create_parser("parse_inlines_nbsp",
7690 function()
7691 return parsers.inlines_nbsp
7692 end, false)
7693 self.create_parser("parse_inlines_no_link_or_emphasis",
7694 function()
7695 return parsers.inlines_no_link_or_emphasis
7696 end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

7697 parsers.minimally_indented_blankline = parsers.check_minimal_indent * (parsers.blankline)
7698
7699 parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")
7700
7701 parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("Block|Paragraph")
7702
7703 parsers.minimally_indented_paragraph = parsers.check_minimal_indent * V("Paragraph")
7704
7705 parsers.minimally_indented_plain = parsers.check_minimal_indent * V("Plain")
7706
7707 parsers.minimally_indented_par_or_plain = parsers.minimally_indented_paragraph
7708 + parsers.minimally_indented_plain
7709
7710 parsers.minimally_indented_par_or_plain_no_blank = parsers.minimally_indented_par_or_plain
7711 - parsers.minimally_indented_blankline
7712
7713 parsers.minimally_indented_ref = parsers.check_minimal_indent * V("Reference")
7714
7715 parsers.minimally_indented_blank = parsers.check_minimal_indent * V("Blank")
7716

```



```

7717 parsers.conditionally_indented_blankline = parsers.check_minimal_blank_indent * (pa
7718
7719 parsers.minimally_indented_ref_or_block = parsers.minimally_indented_ref
7720 + parsers.minimally_indented_block
7721 - parsers.minimally_indented_blankline
7722
7723 parsers.minimally_indented_ref_or_block_or_par = parsers.minimally_indented_ref
7724 + parsers.minimally_indented_block
7725 - parsers.minimally_indented_blankl
7726

```

The following pattern parses the properly indented content that follows the initial container start.

```

7727
7728 parsers.separator_loop = function(separated_block, paragraph, block_separator, para
7729 return separated_block
7730 + block_separator
7731 * paragraph
7732 * separated_block
7733 + paragraph_separator
7734 * paragraph
7735 end
7736
7737 parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
7738 return {
7739 block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
7740 par = parsers.separator_loop(separated_block, paragraph, block_separator, para
7741 }
7742 end
7743
7744 parsers.block_sep_group = function(blank)
7745 return blank^0 * parsers.eof
7746 + (blank^2 / writer.paragraphsep
7747 + blank^0 / writer.interblocksep
7748)
7749 end
7750
7751 parsers.par_sep_group = function(blank)
7752 return blank^0 * parsers.eof
7753 + blank^0 / writer.paragraphsep
7754 end
7755
7756 parsers.sep_group_no_output = function(blank)
7757 return blank^0 * parsers.eof
7758 + blank^0
7759 end
7760

```

```

7761 parsers.content_blank = parsers.minimally_indented_blankline
7762
7763 parsers.ref_or_block_separated = parsers.sep_group_no_output(parsers.content_blank
7764 * (parsers.minimally_indented_ref
7765 - parsers.content_blank)
7766 + parsers.block_sep_group(parsers.content_blank)
7767 * (parsers.minimally_indented_block
7768 - parsers.content_blank)
7769
7770 parsers.loop_body_pair =
7771 parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7772 parsers.minimally_indented_par_or_plain_no_blank,
7773 parsers.block_sep_group(parsers.content_blank),
7774 parsers.par_sep_group(parsers.content_blank))
7775
7776 parsers.content_loop = (V("Block")
7777 * parsers.loop_body_pair.block~0
7778 + (V("Paragraph") + V("Plain"))
7779 * parsers.ref_or_block_separated
7780 * parsers.loop_body_pair.block~0
7781 + (V("Paragraph") + V("Plain"))
7782 * parsers.loop_body_pair.par~0
7783 * parsers.content_blank~0
7784
7785 parsers.indented_content = function()
7786 return Ct((V("Reference") + (parsers.blankline / ""))
7787 * parsers.content_blank~0
7788 * parsers.check_minimal_indent
7789 * parsers.content_loop
7790 + (V("Reference") + (parsers.blankline / ""))
7791 * parsers.content_blank~0
7792 + parsers.content_loop)
7793 end
7794
7795 parsers.add_indent = function(pattern, name, breakable)
7796 return Cg(Cmt(Cb("indent_info")
7797 * Ct(pattern)
7798 * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
7799 * Cc(name)
7800 * Cc(breakable),
7801 process_starter_indent), "indent_info")
7802 end
7803

```

### 3.1.6.4 Parsers Used for Markdown Lists (local)

```

7804 if options.hashEnumerators then

```

```

7805 parsers.dig = parsers.digit + parsers.hash
7806 else
7807 parsers.dig = parsers.digit
7808 end
7809
7810 parsers.enumerator = function(delimiter_type, interrupting)
7811 local delimiter_range
7812 local allowed_end
7813 if interrupting then
7814 delimiter_range = P("1")
7815 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7816 else
7817 delimiter_range = parsers.dig * parsers.dig^-8
7818 allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7819 end
7820
7821 return parsers.check_trail
7822 * Ct(C(delimiter_range) * C(delimiter_type))
7823 * allowed_end
7824 end
7825
7826 parsers.starter = parsers.bullet(parsers.dash)
7827 + parsers.bullet(parsers.asterisk)
7828 + parsers.bullet(parsers.plus)
7829 + parsers.enumerator(parsers.period)
7830 + parsers.enumerator(parsers.rparent)
7831

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

7832 parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.spacec
7833
7834 parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", true)
7835 * parsers.indented_content()
7836 * remove_indent("bq")
7837
7838 if not options.breakableBlockquotes then
7839 parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", fals
7840 * parsers.indented_content()
7841 * remove_indent("bq")
7842 end

```

### 3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```

7843 local function parse_content_part(content_part)
7844 local rope = util.rope_to_string(content_part)

```

```

7845 local parsed = self.parser_functions.parse_inlines_no_link_or_emphasis(ropo)
7846 parsed.indent_info = nil
7847 return parsed
7848 end
7849

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

7850 local function collect_emphasis_content(t, opening_index, closing_index)
7851 local content = {}
7852
7853 local content_part = {}
7854 for i = opening_index, closing_index do
7855 local value = t[i]
7856
7857 if value.rendered ~= nil then
7858 content[#content + 1] = parse_content_part(content_part)
7859 content_part = {}
7860 content[#content + 1] = value.rendered
7861 value.rendered = nil
7862 else
7863 if value.type == "delimiter" and value.element == "emphasis" then
7864 if value.is_active then
7865 content_part[#content_part + 1] = string.rep(value.character, value.curren
7866 end
7867 else
7868 content_part[#content_part + 1] = value.content
7869 end
7870 value.content = ''
7871 value.is_active = false
7872 end
7873 end
7874
7875 if next(content_part) ~= nil then
7876 content[#content + 1] = parse_content_part(content_part)
7877 end
7878
7879 return content
7880 end
7881

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

7882 local function fill_emph(t, opening_index, closing_index)
7883 local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7884 t[opening_index + 1].is_active = true
7885 t[opening_index + 1].rendered = writer.emphasis(content)
7886 end

```

7887

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
7888 local function fill_strong(t, opening_index, closing_index)
7889 local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7890 t[opening_index + 1].is_active = true
7891 t[opening_index + 1].rendered = writer.strong(content)
7892 end
7893
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
7894 local function breaks_three_rule(opening_delimiter, closing_delimiter)
7895 return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
7896 ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0)
7897 and (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count % 3 ~= 0)
7898 end
7899
```

Look for the first potential emphasis opener in the delimiter table `t` in the range from `bottom_index` to `latest_index` that has the same character `character` as the closing delimiter `closing_delimiter`.

```
7900 local function find_emphasis_opener(t, bottom_index, latest_index, character, closing_delimiter)
7901 for i = latest_index, bottom_index, -1 do
7902 local value = t[i]
7903 if value.is_active and
7904 value.is_opening and
7905 value.type == "delimiter" and
7906 value.element == "emphasis" and
7907 (value.character == character) and
7908 (value.current_count > 0) then
7909 if not breaks_three_rule(value, closing_delimiter) then
7910 return i
7911 end
7912 end
7913 end
7914 end
7915
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```
7916 local function process_emphasis(t, opening_index, closing_index)
7917 for i = opening_index, closing_index do
7918 local value = t[i]
7919 if value.type == "delimiter" and value.element == "emphasis" then
7920 local delimiter_length = string.len(value.content)
7921 value.character = string.sub(value.content, 1, 1)
7922 end
7923 end
7924 end
```

```

7922 value.current_count = delimiter_length
7923 value.original_count = delimiter_length
7924 end
7925 end
7926
7927 local openers_bottom = {
7928 ['*'] = {
7929 [true] = {opening_index, opening_index, opening_index},
7930 [false] = {opening_index, opening_index, opening_index}
7931 },
7932 ['_'] = {
7933 [true] = {opening_index, opening_index, opening_index},
7934 [false] = {opening_index, opening_index, opening_index}
7935 }
7936 }
7937
7938 local current_position = opening_index
7939 local max_position = closing_index
7940
7941 while current_position <= max_position do
7942 local value = t[current_position]
7943
7944 if value.type ~= "delimiter" or
7945 value.element ~= "emphasis" or
7946 not value.is_active or
7947 not value.is_closing or
7948 (value.current_count <= 0) then
7949 current_position = current_position + 1
7950 goto continue
7951 end
7952
7953 local character = value.character
7954 local is_opening = value.is_opening
7955 local closing_length_modulo_three = value.original_count % 3
7956
7957 local current_openers_bottom = openers_bottom[character][is_opening][closing_length_modulo_three]
7958
7959 local opener_position = find_emphasis_opener(t, current_openers_bottom, current_position)
7960
7961 if (opener_position == nil) then
7962 current_openers_bottom[character][is_opening][closing_length_modulo_three + 1] = current_position
7963 current_position = current_position + 1
7964 goto continue
7965 end
7966
7967 local opening_delimiter = t[opener_position]
7968

```

```

7969 local current_opening_count = opening_delimiter.current_count
7970 local current_closing_count = t[current_position].current_count
7971
7972 if (current_opening_count >= 2) and (current_closing_count >= 2) then
7973 opening_delimiter.current_count = current_opening_count - 2
7974 t[current_position].current_count = current_closing_count - 2
7975 fill_strong(t, opener_position, current_position)
7976 else
7977 opening_delimiter.current_count = current_opening_count - 1
7978 t[current_position].current_count = current_closing_count - 1
7979 fill_emph(t, opener_position, current_position)
7980 end
7981
7982 ::continue::
7983 end
7984 end
7985
7986 local cont = lpeg.R("\128\191") -- continuation byte
7987

```

Match a UTF-8 character of byte length `n`.

```

7988 local function utf8_by_byte_count(n)
7989 if (n == 1) then
7990 return lpeg.R("\0\127")
7991 end
7992 if (n == 2) then
7993 return lpeg.R("\194\223") * cont
7994 end
7995 if (n == 3) then
7996 return lpeg.R("\224\239") * cont * cont
7997 end
7998 if (n == 4) then
7999 return lpeg.R("\240\244") * cont * cont * cont
8000 end
8001 end

```

Check if there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```

8002 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
8003 local c
8004 local char_length
8005 for pos = start_pos, end_pos, 1 do
8006 if (start_pos < 0) then
8007 char_length = -pos
8008 else
8009 char_length = pos + 1
8010 end
8011

```

```

8012 if (chartype == "punctuation") then
8013 if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8014 return i
8015 end
8016 else
8017 c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
8018 if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8019 return i
8020 end
8021 end
8022 end
8023 end
8024
8025 local function check_preceding_unicode_punctuation(s, i)
8026 return check_unicode_type(s, i, -4, -1, "punctuation")
8027 end
8028
8029 local function check_preceding_unicode_whitespace(s, i)
8030 return check_unicode_type(s, i, -4, -1, "%s")
8031 end
8032
8033 local function check_following_unicode_punctuation(s, i)
8034 return check_unicode_type(s, i, 0, 3, "punctuation")
8035 end
8036
8037 local function check_following_unicode_whitespace(s, i)
8038 return check_unicode_type(s, i, 0, 3, "%s")
8039 end
8040
8041 parsers.unicode_preceding_punctuation = B(parsers.escapable)
8042 + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8043
8044 parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8045
8046 parsers.unicode_following_punctuation = #parsers.escapable
8047 + Cmt(parsers.succeed, check_following_unicode_punctuation)
8048
8049 parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_unicode_whitespace)
8050
8051 parsers.delimiter_run = function(character)
8052 return (B(parsers.backslash * character) + -B(character))
8053 * character^1
8054 * -#character
8055 end
8056
8057 parsers.left_flanking_delimiter_run = function(character)
8058 return (B(parsers.any)

```



```

8059 * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_wh
8060 + -B(parsers.any))
8061 * parsers.delimiter_run(character)
8062 * parsers.unicode_following_punctuation
8063 + parsers.delimiter_run(character)
8064 * -(parsers.unicode_following_punctuation + parsers.unicode_following_wh
8065 + parsers.eof)
8066 end
8067
8068 parsers.right_flanking_delimiter_run = function(character)
8069 return parsers.unicode_preceding_punctuation
8070 * parsers.delimiter_run(character)
8071 * (parsers.unicode_following_punctuation + parsers.unicode_following_whites
8072 + parsers.eof)
8073 + (B(parsers.any)
8074 * -(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whi
8075 * parsers.delimiter_run(character)
8076 end
8077
8078 if options.underscores then
8079 parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8080 + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
8081 + (parsers.unicode_preceding_punctuation
8082 * #parsers.right_flanking_delimiter_run(parsers.underscor
8083 * parsers.left_flanking_delimiter_run(parsers.underscore)
8084
8085 parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8086 + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
8087 + #(parsers.left_flanking_delimiter_run(parsers.underscore)
8088 * parsers.unicode_following_punctuation))
8089 * parsers.right_flanking_delimiter_run(parsers.underscore)
8090 else
8091 parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8092
8093 parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8094 end
8095
8096 parsers.emph_capturing_open_and_close = #parsers.emph_start * #parsers.emph_end
8097 * Ct(Cg(Cc("delimiter"), "type")
8098 * Cg(Cc("emphasis"), "element")
8099 * Cg(C(parsers.emph_start), "content")
8100 * Cg(Cc(true), "is_opening")
8101 * Cg(Cc(true), "is_closing"))
8102
8103 parsers.emph_capturing_open = Ct(Cg(Cc("delimiter"), "type")
8104 * Cg(Cc("emphasis"), "element")
8105 * Cg(C(parsers.emph_start), "content")

```

```

8106 * Cg(Cc(true), "is_opening")
8107 * Cg(Cc(false), "is_closing"))
8108
8109 parsers.emph_capturing_close = Ct(Cg(Cc("delimiter"), "type")
8110 * Cg(Cc("emphasis"), "element")
8111 * Cg(C(parsers.emph_end), "content")
8112 * Cg(Cc(false), "is_opening")
8113 * Cg(Cc(true), "is_closing"))
8114
8115 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8116 + parsers.emph_capturing_open
8117 + parsers.emph_capturing_close
8118
8119 parsers.emph_open = parsers.emph_capturing_open_and_close
8120 + parsers.emph_capturing_open
8121
8122 parsers.emph_close = parsers.emph_capturing_open_and_close
8123 + parsers.emph_capturing_close
8124

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

8125 -- List of references defined in the document
8126 local references
8127

```

The `reader->register_link` method registers a link reference, where `tag` is the link label, `url` is the link destination, `title` is the optional link title, and `attributes` are the optional attributes.

```

8128 function self.register_link(_, tag, url, title,
8129 attributes)
8130 local normalized_tag = self.normalize_tag(tag)
8131 if references[normalized_tag] == nil then
8132 references[normalized_tag] = {
8133 url = url,
8134 title = title,
8135 attributes = attributes
8136 }
8137 end
8138 return ""
8139 end
8140

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

8141 function self.lookup_reference(tag)
8142 return references[self.normalize_tag(tag)]
8143 end
8144

```

```

8145 parsers.title_s_direct_ref = parsers.squote
8146 * Cs((parsers.html_entities
8147 + (parsers.anyescaped - parsers.squote - parsers.bl
8148 * parsers.squote
8149
8150 parsers.title_d_direct_ref = parsers.dquote
8151 * Cs((parsers.html_entities
8152 + (parsers.anyescaped - parsers.dquote - parsers.bl
8153 * parsers.dquote
8154
8155 parsers.title_p_direct_ref = parsers.lparent
8156 * Cs((parsers.html_entities
8157 + (parsers.anyescaped - parsers.lparent - parsers.r
8158 * parsers.rparent
8159
8160 parsers.title_direct_ref = parsers.title_s_direct_ref
8161 + parsers.title_d_direct_ref
8162 + parsers.title_p_direct_ref
8163
8164 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8165 * Cg(parsers.url + Cc(""), "url")
8166 * parsers.spnl
8167 * Cg(parsers.title_direct_ref + Cc(""), "title")
8168 * parsers.spnl * parsers.rparent
8169
8170 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8171 * Cg(parsers.url + Cc(""), "url")
8172 * parsers.spnlc
8173 * Cg(parsers.title + Cc(""), "title")
8174 * parsers.spnlc * parsers.rparent
8175
8176 parsers.empty_link = parsers.lbracket
8177 * parsers.rbracket
8178
8179 parsers.inline_link = parsers.link_text
8180 * parsers.inline_direct_ref
8181
8182 parsers.full_link = parsers.link_text
8183 * parsers.link_label
8184
8185 parsers.shortcut_link = parsers.link_label
8186 * -(parsers.empty_link + parsers.link_label)
8187
8188 parsers.collapsed_link = parsers.link_label
8189 * parsers.empty_link
8190
8191 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)

```

```

8192 * Cg(Cc("inline"), "link_type")
8193 + #(parsers.exclamation * parsers.full_link)
8194 * Cg(Cc("full"), "link_type")
8195 + #(parsers.exclamation * parsers.collapsed_link)
8196 * Cg(Cc("collapsed"), "link_type")
8197 + #(parsers.exclamation * parsers.shortcut_link)
8198 * Cg(Cc("shortcut"), "link_type")
8199 + #(parsers.exclamation * parsers.empty_link)
8200 * Cg(Cc("empty"), "link_type")
8201
8202 parsers.link_opening = #parsers.inline_link
8203 * Cg(Cc("inline"), "link_type")
8204 + #parsers.full_link
8205 * Cg(Cc("full"), "link_type")
8206 + #parsers.collapsed_link
8207 * Cg(Cc("collapsed"), "link_type")
8208 + #parsers.shortcut_link
8209 * Cg(Cc("shortcut"), "link_type")
8210 + #parsers.empty_link
8211 * Cg(Cc("empty_link"), "link_type")
8212 + #parsers.link_text
8213 * Cg(Cc("link_text"), "link_type")
8214
8215 parsers.link_image_opening = Ct(Cg(Cc("delimiter"), "type")
8216 * Cg(Cc(true), "is_opening")
8217 * Cg(Cc(false), "is_closing")
8218 * (Cg(Cc("image"), "element")
8219 * parsers.image_opening
8220 * Cg(parsers.exclamation * parsers.lbracket, "content")
8221 + Cg(Cc("link"), "element")
8222 * parsers.link_opening
8223 * Cg(parsers.lbracket, "content")))
8224
8225 parsers.link_image_closing = Ct(Cg(Cc("delimiter"), "type")
8226 * Cg(Cc("link"), "element")
8227 * Cg(Cc(false), "is_opening")
8228 * Cg(Cc(true), "is_closing")
8229 * (Cg(Cc(true), "is_direct")
8230 * Cg(parsers.rbracket * #parsers.inline_direct_re
8231 + Cg(Cc(false), "is_direct")
8232 * Cg(parsers.rbracket, "content")))
8233
8234 parsers.link_image_open_or_close = parsers.link_image_opening
8235 + parsers.link_image_closing
8236
8237 if options.html then
8238 parsers.link_emph_precedence = parsers.inticks

```

```

8239 + parsers.autolink
8240 + parsers.html_inline_tags
8241 else
8242 parsers.link_emph_precedence = parsers.inticks
8243 + parsers.autolink
8244 end
8245
8246 parsers.link_and_emph_endline = parsers.newline
8247 * ((parsers.check_minimal_indent
8248 * -V("EndlineExceptions")
8249 + parsers.check_optional_indent
8250 * -V("EndlineExceptions")
8251 * -parsers.starter) / "")
8252 * parsers.spacechar^0 / "\n"
8253
8254 parsers.link_and_emph_content = Ct(Cg(Cc("content"), "type")
8255 * Cg(Cs((parsers.link_emph_precedence
8256 + parsers.backslash * parsers.any
8257 + parsers.link_and_emph_endline
8258 + (parsers.linechar
8259 - parsers.blankline^2
8260 - parsers.link_image_open_or_close
8261 - parsers.emph_open_or_close))^0), "con
8262
8263 parsers.link_and_emph_table = (parsers.link_image_opening + parsers.emph_open)
8264 * parsers.link_and_emph_content
8265 * ((parsers.link_image_open_or_close + parsers.emph_ope
8266 * parsers.link_and_emph_content)^1
8267

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8268 local function collect_link_content(t, opening_index, closing_index)
8269 local content = {}
8270 for i = opening_index, closing_index do
8271 content[#content + 1] = t[i].content
8272 end
8273 return util.ropo_to_string(content)
8274 end
8275

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

8276 local function find_link_opener(t, bottom_index, latest_index)
8277 for i = latest_index, bottom_index, -1 do
8278 local value = t[i]
8279 if value.type == "delimiter" and
8280 value.is_opening and

```

```

8281 (value.element == "link" or value.element == "image")
8282 and not value.removed then
8283 if value.is_active then
8284 return i
8285 end
8286 value.removed = true
8287 return nil
8288 end
8289 end
8290 end
8291

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

8292 local function find_next_link_closing_index(t, latest_index)
8293 for i = latest_index, #t do
8294 local value = t[i]
8295 if value.is_closing and
8296 value.element == "link" and
8297 not value.removed then
8298 return i
8299 end
8300 end
8301 end
8302

```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```

8303 local function disable_previous_link_openers(t, opening_index)
8304 if t[opening_index].element == "image" then
8305 return
8306 end
8307
8308 for i = opening_index, 1, -1 do
8309 local value = t[i]
8310 if value.is_active and
8311 value.type == "delimiter" and
8312 value.is_opening and
8313 value.element == "link" then
8314 value.is_active = false
8315 end
8316 end
8317 end
8318

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

8319 local function disable_range(t, opening_index, closing_index)

```

```

8320 for i = opening_index, closing_index do
8321 local value = t[i]
8322 if value.is_active then
8323 value.is_active = false
8324 if value.type == "delimiter" then
8325 value.removed = true
8326 end
8327 end
8328 end
8329 end
8330

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8331 local function delete_parsed_content_in_range(t, opening_index, closing_index)
8332 for i = opening_index, closing_index do
8333 t[i].rendered = nil
8334 end
8335 end
8336

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8337 local function empty_content_in_range(t, opening_index, closing_index)
8338 for i = opening_index, closing_index do
8339 t[i].content = ''
8340 end
8341 end
8342

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

8343 local function join_attributes(reference_attributes, own_attributes)
8344 local merged_attributes = {}
8345 for _, attribute in ipairs(reference_attributes or {}) do
8346 table.insert(merged_attributes, attribute)
8347 end
8348 for _, attribute in ipairs(own_attributes or {}) do
8349 table.insert(merged_attributes, attribute)
8350 end
8351 if next(merged_attributes) == nil then
8352 merged_attributes = nil
8353 end
8354 return merged_attributes
8355 end
8356

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
8357 local function render_link_or_image(t, opening_index, closing_index, content_end_index)
8358 process_emphasis(t, opening_index, content_end_index)
8359 local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index - 1)
8360
8361 local rendered = {}
8362 if (t[opening_index].element == "link") then
8363 rendered = writer.link(mapped, reference.url, reference.title, reference.attributes)
8364 end
8365
8366 if (t[opening_index].element == "image") then
8367 rendered = writer.image(mapped, reference.url, reference.title, reference.attributes)
8368 end
8369
8370 t[opening_index].rendered = rendered
8371 delete_parsed_content_in_range(t, opening_index + 1, closing_index)
8372 empty_content_in_range(t, opening_index, closing_index)
8373 disable_previous_link_openers(t, opening_index)
8374 disable_range(t, opening_index, closing_index)
8375 end
8376
```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```
8377 local function resolve_inline_following_content(t, closing_index, match_reference,
8378 local content = ""
8379 for i = closing_index + 1, #t do
8380 content = content .. t[i].content
8381 end
8382
8383 local matching_content = parsers.succeed
8384
8385 if match_reference then
8386 matching_content = matching_content * parsers.inline_direct_ref_inside
8387 end
8388
8389 if match_link_attributes then
8390 matching_content = matching_content * Cg(Ct(parsers.attributes^1), "attributes")
8391 end
8392
8393 local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), content)
8394
8395 local matched_count = matched.end_position - 1
8396 for i = closing_index + 1, #t do
```



```

8397 local value = t[i]
8398
8399 local chars_left = matched_count
8400 matched_count = matched_count - #value.content
8401
8402 if matched_count <= 0 then
8403 value.content = value.content:sub(chars_left + 1)
8404 break
8405 end
8406
8407 value.content = ''
8408 value.is_active = false
8409 end
8410
8411 local attributes = matched.attributes
8412 if attributes == nil or next(attributes) == nil then
8413 attributes = nil
8414 end
8415
8416 return {
8417 url = matched.url or "",
8418 title = matched.title or "",
8419 attributes = attributes
8420 }
8421 end
8422

```

Resolve an inline link a<sup>33</sup> from the delimiters at [opening\\_index](#) and [closing\\_index](#) within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```

8423 local function resolve_inline_link(t, opening_index, closing_index)
8424 local inline_content = resolve_inline_following_content(t, closing_index, true, t)
8425 render_link_or_image(t, opening_index, closing_index, closing_index, inline_content)
8426 end
8427

```

Resolve a shortcut link [a] from the delimiters at [opening\\_index](#) and [closing\\_index](#) within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

8428 local function resolve_shortcut_link(t, opening_index, closing_index)
8429 local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8430 local r = self.lookup_reference(content)
8431
8432 if r then
8433 local inline_content = resolve_inline_following_content(t, closing_index, false, r)
8434 r.attributes = join_attributes(r.attributes, inline_content.attributes)
8435 render_link_or_image(t, opening_index, closing_index, closing_index, r)

```

---

<sup>33</sup>See [b](#).

```

8436 end
8437 end
8438

```

Resolve a full link [a][b] from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `b` is not found in the references.

```

8439 local function resolve_full_link(t, opening_index, closing_index)
8440 local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8441 local next_link_content = collect_link_content(t, closing_index + 3, next_link_closing_index)
8442 local r = self.lookup_reference(next_link_content)
8443
8444 if r then
8445 local inline_content = resolve_inline_following_content(t, next_link_closing_index,
8446 t.match_link_attributes)
8447 r.attributes = join_attributes(r.attributes, inline_content.attributes)
8448 render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8449 r)
8449 end
8450 end
8451

```

Resolve a collapsed link [a] from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

8452 local function resolve_collapsed_link(t, opening_index, closing_index)
8453 local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8454 local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8455 local r = self.lookup_reference(content)
8456
8457 if r then
8458 local inline_content = resolve_inline_following_content(t, closing_index, false)
8459 r.attributes = join_attributes(r.attributes, inline_content.attributes)
8460 render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8461 r)
8461 end
8462 end
8463

```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```

8464 local function process_links_and_emphasis(t)
8465 for _,value in ipairs(t) do
8466 value.is_active = true
8467 end
8468
8469 for i,value in ipairs(t) do
8470 if not value.is_closing or
8471 value.type ~= "delimiter" or

```

```

8472 not (value.element == "link" or value.element == "image") then
8473 goto continue
8474 end
8475
8476 local opener_position = find_link_opener(t, 1, i - 1)
8477 if (opener_position == nil) then
8478 goto continue
8479 end
8480
8481 local opening_delimiter = t[opener_position]
8482 opening_delimiter.removed = true
8483
8484 local link_type = opening_delimiter.link_type
8485
8486 if (link_type == "inline") then
8487 resolve_inline_link(t, opener_position, i)
8488 end
8489 if (link_type == "shortcut") then
8490 resolve_shortcut_link(t, opener_position, i)
8491 end
8492 if (link_type == "full") then
8493 resolve_full_link(t, opener_position, i)
8494 end
8495 if (link_type == "collapsed") then
8496 resolve_collapsed_link(t, opener_position, i)
8497 end
8498
8499 ::continue::
8500 end
8501
8502 t[#t].content = t[#t].content:gsub("%s*$", "")
8503
8504 process_emphasis(t, 1, #t)
8505 local final_result = collect_emphasis_content(t, 1, #t)
8506 return final_result
8507 end
8508
8509 function self.defer_link_and_emphasis_processing(delimiter_table)
8510 return writer.defer_call(function()
8511 return process_links_and_emphasis(delimiter_table)
8512 end)
8513 end
8514

```

### 3.1.6.8 Inline Elements (local)

```

8515 parsers.Str = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)

```

```

8516 / writer.string
8517
8518 parsers.Symbol = (parsers.backtick^1 + V("SpecialChar"))
8519 / writer.string
8520
8521 parsers.Ellipsis = P("...") / writer.ellipsis
8522
8523 parsers.Smart = parsers.Ellipsis
8524
8525 parsers.Code = parsers.inticks / writer.code
8526
8527 if options.blankBeforeBlockquote then
8528 parsers.bqstart = parsers.fail
8529 else
8530 parsers.bqstart = parsers.blockquote_start
8531 end
8532
8533 if options.blankBeforeHeading then
8534 parsers.headerstart = parsers.fail
8535 else
8536 parsers.headerstart = parsers.atx_heading
8537 end
8538
8539 if options.blankBeforeList then
8540 parsers.interrupting_bullets = parsers.fail
8541 parsers.interrupting_enumerators = parsers.fail
8542 else
8543 parsers.interrupting_bullets = parsers.bullet(parsers.dash, true)
8544 + parsers.bullet(parsers.asterisk, true)
8545 + parsers.bullet(parsers.plus, true)
8546
8547 parsers.interrupting_enumerators = parsers.enumerator(parsers.period, true)
8548 + parsers.enumerator(parsers.rparent, true)
8549 end
8550
8551 if options.html then
8552 parsers.html_interrupting = parsers.check_trail
8553 * (parsers.html_incomplete_open_tag
8554 + parsers.html_incomplete_close_tag
8555 + parsers.html_incomplete_open_special_tag
8556 + parsers.html_comment_start
8557 + parsers.html_cdatasection_start
8558 + parsers.html_declaration_start
8559 + parsers.html_instruction_start
8560 - parsers.html_close_special_tag
8561 - parsers.html_empty_special_tag)
8562 else

```

```

8563 parsers.html_interrupting = parsers.fail
8564 end
8565
8566 parsers.EndlineExceptions
8567 = parsers.blankline -- paragraph break
8568 + parsers.eof -- end of document
8569 + parsers.bqstart
8570 + parsers.thematic_break_lines
8571 + parsers.interrupting_bullets
8572 + parsers.interrupting_enumerators
8573 + parsers.headerstart
8574 + parsers.html_interrupting
8575
8576 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8577
8578 parsers.endline = parsers.newline
8579 * (parsers.check_minimal_indent
8580 * -V("EndlineExceptions")
8581 + parsers.check_optional_indent
8582 * -V("EndlineExceptions")
8583 * -parsers.starter)
8584 * parsers.spacechar^0
8585
8586 parsers.Endline = parsers.endline
8587 / writer.soft_line_break
8588
8589 parsers.EndlineNoSub = parsers.endline
8590
8591 parsers.NoSoftLineBreakEndline
8592 = parsers.newline
8593 * (parsers.check_minimal_indent
8594 * -V("NoSoftLineBreakEndlineExceptions")
8595 + parsers.check_optional_indent
8596 * -V("NoSoftLineBreakEndlineExceptions")
8597 * -parsers.starter)
8598 * parsers.spacechar^0
8599 / writer.space
8600
8601 parsers.EndlineBreak = parsers.backslash * parsers.Endline
8602 / writer.hard_line_break
8603
8604 parsers.OptionalIndent
8605 = parsers.spacechar^1 / writer.space
8606
8607 parsers.Space = parsers.spacechar^2 * parsers.Endline
8608 / writer.hard_line_break
8609 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.

```

```

8610 + parsers.spacechar^1 * parsers.Endline
8611 / writer.soft_line_break
8612 + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8613
8614 parsers.NoSoftLineBreakSpace
8615 = parsers.spacechar^2 * parsers.Endline
8616 / writer.hard_line_break
8617 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8618 + parsers.spacechar^1 * parsers.Endline
8619 / writer.soft_line_break
8620 + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8621
8622 parsers.NonbreakingEndline
8623 = parsers.endline
8624 / writer.soft_line_break
8625
8626 parsers.NonbreakingSpace
8627 = parsers.spacechar^2 * parsers.Endline
8628 / writer.hard_line_break
8629 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
8630 + parsers.spacechar^1 * parsers.Endline
8631 * parsers.optionalspace
8632 / writer.soft_line_break
8633 + parsers.spacechar^1 * parsers.optionalspace
8634 / writer.nbsp
8635

```

The `reader->auto_link_url` method produces an autolink to a URL or a relative reference in the output format, where `url` is the link destination and `attributes` are the optional attributes.

```

8636 function self.auto_link_url(url, attributes)
8637 return writer.link(writer.escape(url),
8638 url, nil, attributes)
8639 end

```

The `reader->auto_link_email` method produces an autolink to an e-mail in the output format, where `email` is the email address destination and `attributes` are the optional attributes.

```

8640 function self.auto_link_email(email, attributes)
8641 return writer.link(writer.escape(email),
8642 "mailto:".email,
8643 nil, attributes)
8644 end
8645
8646 parsers.AutoLinkUrl = parsers.auto_link_url
8647 / self.auto_link_url
8648

```

```

8649 parsers.AutoLinkEmail
8650 = parsers.auto_link_email
8651 / self.auto_link_email
8652
8653 parsers.AutoLinkRelativeReference
8654 = parsers.auto_link_relative_reference
8655 / self.auto_link_url
8656
8657 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
8658 / self.defer_link_and_emphasis_processing
8659
8660 parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
8661
8662 parsers.InlineHtml = Cs(parsers.html_inline_comment) / writer.inline_html_comment
8663 + Cs(parsers.html_any_empty_inline_tag
8664 + parsers.html_inline_instruction
8665 + parsers.html_inline_cdatasection
8666 + parsers.html_inline_declaration
8667 + parsers.html_any_open_inline_tag
8668 + parsers.html_any_close_tag)
8669 / writer.inline_html_tag
8670
8671 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

8672 parsers.DisplayHtml = Cs(parsers.check_trail
8673 * (parsers.html_comment
8674 + parsers.html_special_block
8675 + parsers.html_block
8676 + parsers.html_any_block
8677 + parsers.html_instruction
8678 + parsers.html_cdatasection
8679 + parsers.html_declaration))
8680 / writer.block_html_element
8681
8682 parsers.indented_non_blank_line = parsers.indentedline - parsers.blankline
8683
8684 parsers.Verbatim = Cs(
8685 parsers.check_code_trail
8686 * (parsers.line - parsers.blankline)
8687 * ((parsers.check_minimal_blank_indent_and_full_code_trail * pa
8688 * ((parsers.check_minimal_indent / "") * parsers.check_code_t
8689 * (parsers.line - parsers.blankline))^1)^0
8690) / self.expandtabs / writer.verbatim
8691
8692 parsers.Blockquote = parsers.blockquote_body

```

```

8693 / writer.blockquote
8694
8695 parsers.ThematicBreak = parsers.thematic_break_lines
8696 / writer.thematic_break
8697
8698 parsers.Reference = parsers.define_reference_parser
8699 / self.register_link
8700
8701 parsers.Paragraph = parsers.freeze_trail
8702 * (Ct((parsers.Inline)^1)
8703 * (parsers.newline + parsers.eof)
8704 * parsers.unfreeze_trail
8705 / writer.paragraph)
8706
8707 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
8708 / writer.plain

```

### 3.1.6.10 Lists (local)

```

8709
8710 if options.taskLists then
8711 parsers.tickbox = (parsers.ticked_box
8712 + parsers.halfticked_box
8713 + parsers.unticked_box
8714) / writer.tickbox
8715 else
8716 parsers.tickbox = parsers.fail
8717 end
8718
8719 parsers.list_blank = parsers.conditionally_indented_blankline
8720
8721 parsers.ref_or_block_list_separated = parsers.sep_group_no_output(parsers.list_blank
8722 * parsers.minimally_indented_ref
8723 + parsers.block_sep_group(parsers.list_blank)
8724 * parsers.minimally_indented_block)
8725
8726 parsers.ref_or_block_non_separated = parsers.minimally_indented_ref
8727 + (parsers.succeed / writer.interblocksep)
8728 * parsers.minimally_indented_block
8729 - parsers.minimally_indented_blankline
8730
8731 parsers.tight_list_loop_body_pair =
8732 parsers.create_loop_body_pair(parsers.ref_or_block_non_separated,
8733 parsers.minimally_indented_par_or_plain_no_blank,
8734 (parsers.succeed / writer.interblocksep),
8735 (parsers.succeed / writer.paragraphsep))
8736

```



```

8737 parsers.loose_list_loop_body_pair =
8738 parsers.create_loop_body_pair(parsers.ref_or_block_list_separated,
8739 parsers.minimally_indented_par_or_plain,
8740 parsers.block_sep_group(parsers.list_blank),
8741 parsers.par_sep_group(parsers.list_blank))
8742
8743 parsers.tight_list_content_loop = V("Block")
8744 * parsers.tight_list_loop_body_pair.block^0
8745 + (V("Paragraph") + V("Plain"))
8746 * parsers.ref_or_block_non_separated
8747 * parsers.tight_list_loop_body_pair.block^0
8748 + (V("Paragraph") + V("Plain"))
8749 * parsers.tight_list_loop_body_pair.par^0
8750
8751 parsers.loose_list_content_loop = V("Block")
8752 * parsers.loose_list_loop_body_pair.block^0
8753 + (V("Paragraph") + V("Plain"))
8754 * parsers.ref_or_block_list_separated
8755 * parsers.loose_list_loop_body_pair.block^0
8756 + (V("Paragraph") + V("Plain"))
8757 * parsers.loose_list_loop_body_pair.par^0
8758
8759 parsers.list_item_tightness_condition = -(parsers.list_blank^0
8760 * parsers.minimally_indented_ref_or_block
8761 * remove_indent("li")
8762 + remove_indent("li")
8763 * parsers.fail
8764
8765 parsers.indented_content_tight = Ct((parsers.blankline / "")
8766 * #parsers.list_blank
8767 * remove_indent("li")
8768 + ((V("Reference") + (parsers.blankline / ""))
8769 * parsers.check_minimal_indent
8770 * parsers.tight_list_content_loop
8771 + (V("Reference") + (parsers.blankline / ""))
8772 + (parsers.tickbox^-1 / writer.escape)
8773 * parsers.tight_list_content_loop
8774)
8775 * parsers.list_item_tightness_condition
8776)
8777
8778 parsers.indented_content_loose = Ct((parsers.blankline / "")
8779 * #parsers.list_blank
8780 + ((V("Reference") + (parsers.blankline / ""))
8781 * parsers.check_minimal_indent
8782 * parsers.loose_list_content_loop
8783 + (V("Reference") + (parsers.blankline / ""))

```

```

8784 + (parsers.tickbox^-1 / writer.escape)
8785 * parsers.loose_list_content_loop
8786)
8787)
8788
8789 parsers.TightListItem = function(starter)
8790 return -parsers.ThematicBreak
8791 * parsers.add_indent(starter, "li")
8792 * parsers.indented_content_tight
8793 end
8794
8795 parsers.LooseListItem = function(starter)
8796 return -parsers.ThematicBreak
8797 * parsers.add_indent(starter, "li")
8798 * parsers.indented_content_loose
8799 * remove_indent("li")
8800 end
8801
8802 parsers.BulletListOfType = function(bullet_type)
8803 local bullet = parsers.bullet(bullet_type)
8804 return (Ct(parsers.TightListItem(bullet)
8805 * ((parsers.check_minimal_indent / "")
8806 * parsers.TightListItem(bullet)
8807)^0
8808)
8809 * Cc(true)
8810 * -#((parsers.list_blank^0 / "")
8811 * parsers.check_minimal_indent
8812 * (bullet - parsers.ThematicBreak)
8813)
8814 + Ct(parsers.LooseListItem(bullet)
8815 * ((parsers.list_blank^0 / "")
8816 * (parsers.check_minimal_indent / "")
8817 * parsers.LooseListItem(bullet)
8818)^0
8819)
8820 * Cc(false)
8821) / writer.bulletlist
8822 end
8823
8824 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
8825 + parsers.BulletListOfType(parsers.asterisk)
8826 + parsers.BulletListOfType(parsers.plus)
8827
8828 local function ordered_list(items,tight,starter)
8829 local startnum = starter[2][1]
8830 if options.startNumber then

```

```

8831 startnum = tonumber(startnum) or 1 -- fallback for '#'
8832 if startnum ~= nil then
8833 startnum = math.floor(startnum)
8834 end
8835 else
8836 startnum = nil
8837 end
8838 return writer.orderedlist(items,tight,startnum)
8839 end
8840
8841 parsers.OrderedListOfType = function(delimiter_type)
8842 local enumerator = parsers.enumerator(delimiter_type)
8843 return Cg(enumerator, "listtype")
8844 * (Ct(parsers.TightListItem(Cb("listtype"))
8845 * ((parsers.check_minimal_indent / "") * parsers.TightListItem(enumerat
8846 * Cc(true)
8847 * -#((parsers.list_blank^0 / ""))
8848 * parsers.check_minimal_indent * enumerator)
8849 + Ct(parsers.LooseListItem(Cb("listtype"))
8850 * ((parsers.list_blank^0 / ""))
8851 * (parsers.check_minimal_indent / "") * parsers.LooseListItem(enumerat
8852 * Cc(false)
8853) * Ct(Cb("listtype"))) / ordered_list
8854 end
8855
8856 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
8857 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

8858 parsers.Blank = parsers.blankline / ""
8859 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

8860 function parsers.parse_heading_text(s)
8861 local inlines = self.parser_functions.parse_inlines(s)
8862 local flatten_inlines = self.writer.flatten_inlines
8863 self.writer.flatten_inlines = true
8864 local flat_text = self.parser_functions.parse_inlines(s)
8865 flat_text = util.rope_to_string(flat_text)
8866 self.writer.flatten_inlines = flatten_inlines
8867 return {flat_text, inlines}
8868 end
8869
8870 -- parse atx header
8871 parsers.AtxHeading = parsers.check_trail_no_rem
8872 * Cg(parsers.heading_start, "level")

```

```

8873 * ((C(parsers.optionalspace
8874 * parsers.hash^0
8875 * parsers.optionalspace
8876 * parsers.newline)
8877 + parsers.spacechar^1
8878 * C(parsers.line))
8879 / strip_atx_end
8880 / parsers.parse_heading_text)
8881 * Cb("level")
8882 / writer.heading
8883
8884 parsers.heading_line = parsers.linechar^1
8885 - parsers.thematic_break_lines
8886
8887 parsers.heading_text = parsers.heading_line
8888 * ((V("Endline") / "\n") * (parsers.heading_line - parsers.heading_line)
8889 * parsers.newline^-1)
8890
8891 parsers.SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
8892 * #(parsers.heading_text
8893 * parsers.check_minimal_indent * parsers.check_trail * parsers.newline)
8894 * Cs(parsers.heading_text)
8895 / parsers.parse_heading_text
8896 * parsers.check_minimal_indent_and_trail * parsers.heading_line
8897 * parsers.newline
8898 * parsers.unfreeze_trail
8899 / writer.heading
8900
8901 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain  $\text{T}_{\text{E}}\text{X}$  output.

```

8902 function self.finalize_grammar(extensions)

```

Create a local writable copy of the global read-only `walkable_syntax` hash table. This table can be used by user-defined syntax extensions to insert new PEG patterns into existing rules of the PEG grammar of markdown using the `reader->insert_pattern` method. Furthermore, built-in syntax extensions can use this table to override existing rules using the `reader->update_rule` method.

```

8903 local walkable_syntax = (function(global_walkable_syntax)
8904 local local_walkable_syntax = {}
8905 for lhs, rule in pairs(global_walkable_syntax) do
8906 local_walkable_syntax[lhs] = util.table_copy(rule)
8907 end

```

```

8908 return local_walkable_syntax
8909 end)(walkable_syntax)

```

The `reader->insert_pattern` method adds a pattern to `walkable_syntax` [*left-hand side terminal symbol*] before, instead of, or after a right-hand-side terminal symbol.

```

8910 local current_extension_name = nil
8911 self.insert_pattern = function(selector, pattern, pattern_name)
8912 assert(pattern_name == nil or type(pattern_name) == "string")
8913 local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
8914 assert(lhs ~= nil,
8915 [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
8916 .. selector .. [[]])
8917 assert(walkable_syntax[lhs] ~= nil,
8918 [[Rule]] .. lhs .. [[-> ... does not exist in markdown grammar]])
8919 assert(pos == "before" or pos == "after" or pos == "instead of",
8920 [[Expected positional specifier "before", "after", or "instead of", not "]]
8921 .. pos .. [[]])
8922 local rule = walkable_syntax[lhs]
8923 local index = nil
8924 for current_index, current_rhs in ipairs(rule) do
8925 if type(current_rhs) == "string" and current_rhs == rhs then
8926 index = current_index
8927 if pos == "after" then
8928 index = index + 1
8929 end
8930 break
8931 end
8932 end
8933 assert(index ~= nil,
8934 [[Rule]] .. lhs .. [[->]] .. rhs
8935 .. [[does not exist in markdown grammar]])
8936 local accountable_pattern
8937 if current_extension_name then
8938 accountable_pattern = { pattern, current_extension_name, pattern_name }
8939 else
8940 assert(type(pattern) == "string",
8941 [[reader->insert_pattern() was called outside an extension with]]
8942 .. [[a PEG pattern instead of a rule name]])
8943 accountable_pattern = pattern
8944 end
8945 if pos == "instead of" then
8946 rule[index] = accountable_pattern
8947 else
8948 table.insert(rule, index, accountable_pattern)
8949 end
8950 end

```

Create a local `syntax` hash table that stores those rules of the PEG grammar of markdown that can't be represented as an ordered choice of terminal symbols.

```

8951 local syntax =
8952 { "Blocks",
8953
8954 Blocks = V("InitializeState")
8955 * (V("ExpectedJekyllData")
8956 * (V("Blank")^0 / writer.interblocksep)
8957)^-1
8958 * V("Blank")^0

```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```

8959
8960
8961
8962
8963
8964
8965
8966
8967
8968
8969
8970
8971
8972
8973
8974
8975
8976
8977
8978
8979
8980
8981
8982
8983
8984
8985
8986
8987
8988
8989
8990
8991

```

```

* (V("Block")
 * (V("Blank")^0 * parsers.eof
 + (V("Blank")^2 / writer.paragraphsep
 + V("Blank")^0 / writer.interblocksep
)
)
+ (V("Paragraph") + V("Plain"))
* (V("Blank")^0 * parsers.eof
 + (V("Blank")^2 / writer.paragraphsep
 + V("Blank")^0 / writer.interblocksep
)
)
* V("Block")
* (V("Blank")^0 * parsers.eof
 + (V("Blank")^2 / writer.paragraphsep
 + V("Blank")^0 / writer.interblocksep
)
)
+ (V("Paragraph") + V("Plain"))
* (V("Blank")^0 * parsers.eof
 + V("Blank")^0 / writer.paragraphsep
)
)^0,

ExpectedJekyllData = parsers.fail,

Blank = parsers.Blank,
Reference = parsers.Reference,

Blockquote = parsers.Blockquote,
Verbatim = parsers.Verbatim,
ThematicBreak = parsers.ThematicBreak,
BulletList = parsers.BulletList,

```

```

8992 OrderedList = parsers.OrderedList,
8993 DisplayHtml = parsers.DisplayHtml,
8994 Heading = parsers.Heading,
8995 Paragraph = parsers.Paragraph,
8996 Plain = parsers.Plain,
8997
8998 EndlineExceptions = parsers.EndlineExceptions,
8999 NoSoftLineBreakEndlineExceptions
9000 = parsers.NoSoftLineBreakEndlineExceptions,
9001
9002 Str = parsers.Str,
9003 Space = parsers.Space,
9004 NoSoftLineBreakSpace = parsers.NoSoftLineBreakSpace,
9005 OptionalIndent = parsers.OptionalIndent,
9006 Endline = parsers.Endline,
9007 EndlineNoSub = parsers.EndlineNoSub,
9008 NoSoftLineBreakEndline
9009 = parsers.NoSoftLineBreakEndline,
9010 EndlineBreak = parsers.EndlineBreak,
9011 LinkAndEmph = parsers.LinkAndEmph,
9012 Code = parsers.Code,
9013 AutoLinkUrl = parsers.AutoLinkUrl,
9014 AutoLinkEmail = parsers.AutoLinkEmail,
9015 AutoLinkRelativeReference
9016 = parsers.AutoLinkRelativeReference,
9017 InlineHtml = parsers.InlineHtml,
9018 HtmlEntity = parsers.HtmlEntity,
9019 EscapedChar = parsers.EscapedChar,
9020 Smart = parsers.Smart,
9021 Symbol = parsers.Symbol,
9022 SpecialChar = parsers.fail,
9023 InitializeState = parsers.succeed,
9024 }

```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a function that accepts the current PEG pattern in `walkable_syntax[left-hand side terminal symbol]` if defined or `nil` otherwise and returns a PEG pattern that will (re)define `walkable_syntax[left-hand side terminal symbol]`.

```

9025 self.update_rule = function(rule_name, get_pattern)
9026 assert(current_extension_name ~= nil)
9027 assert(syntax[rule_name] ~= nil,
9028 [[Rule]] .. rule_name .. [[-> ... does not exist in markdown grammar]])
9029 local previous_pattern
9030 local extension_name
9031 if walkable_syntax[rule_name] then
9032 local previous_accountable_pattern = walkable_syntax[rule_name][1]

```

```

9033 previous_pattern = previous_accountable_pattern[1]
9034 extension_name = previous_accountable_pattern[2] .. ", " .. current_extension
9035 else
9036 previous_pattern = nil
9037 extension_name = current_extension_name
9038 end
9039 local pattern

```

Instead of a function, a PEG pattern `pattern` may also be supplied with roughly the same effect as supplying the following function, which will define `walkable_syntax[left-hand side terminal symbol]` unless it has been previously defined.

```

function(previous_pattern)
 assert(previous_pattern == nil)
 return pattern
end

```

```

9040 if type(get_pattern) == "function" then
9041 pattern = get_pattern(previous_pattern)
9042 else
9043 assert(previous_pattern == nil,
9044 [[Rule]] .. rule_name ..
9045 [[has already been updated by]] .. extension_name)
9046 pattern = get_pattern
9047 end
9048 local accountable_pattern = { pattern, extension_name, rule_name }
9049 walkable_syntax[rule_name] = { accountable_pattern }
9050 end

```

Define a hash table of all characters with special meaning and add method `reader->add_special_character` that extends the hash table and updates the PEG grammar of markdown.

```

9051 local special_characters = {}
9052 self.add_special_character = function(c)
9053 table.insert(special_characters, c)
9054 syntax.SpecialChar = S(table.concat(special_characters, ""))
9055 end
9056
9057 self.add_special_character("*")
9058 self.add_special_character("[")
9059 self.add_special_character("]")
9060 self.add_special_character("<")
9061 self.add_special_character("!")
9062 self.add_special_character("\\")

```



Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```
9063 self.initialize_named_group = function(name, value)
9064 local pattern = Ct("")
9065 if value ~= nil then
9066 pattern = pattern / value
9067 end
9068 syntax.InitializeState = syntax.InitializeState
9069 * Cg(pattern, name)
9070 end
```

Add a named group for indentation.

```
9071 self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
9072 for _, extension in ipairs(extensions) do
9073 current_extension_name = extension.name
9074 extension.extend_writer(writer)
9075 extension.extend_reader(self)
9076 end
9077 current_extension_name = nil
```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```
9078 if options.debugExtensions then
9079 local sorted_lhs = {}
9080 for lhs, _ in pairs(walkable_syntax) do
9081 table.insert(sorted_lhs, lhs)
9082 end
9083 table.sort(sorted_lhs)
9084
9085 local output_lines = {"{"}
9086 for lhs_index, lhs in ipairs(sorted_lhs) do
9087 local encoded_lhs = util.encode_json_string(lhs)
9088 table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": []}])
9089 local rule = walkable_syntax[lhs]
9090 for rhs_index, rhs in ipairs(rule) do
9091 local human_readable_rhs
9092 if type(rhs) == "string" then
9093 human_readable_rhs = rhs
9094 else
9095 local pattern_name
9096 if rhs[3] then
9097 pattern_name = rhs[3]
9098 else
9099 pattern_name = "Anonymous Pattern"
9100 end
9101 local extension_name = rhs[2]
```

```

9102 human_readable_rhs = pattern_name .. [[(]] .. extension_name .. [[]]
9103 end
9104 local encoded_rhs = util.encode_json_string(human_readable_rhs)
9105 local output_line = [[]] .. encoded_rhs
9106 if rhs_index < #rule then
9107 output_line = output_line .. ", "
9108 end
9109 table.insert(output_lines, output_line)
9110 end
9111 local output_line = "]"
9112 if lhs_index < #sorted_lhs then
9113 output_line = output_line .. ", "
9114 end
9115 table.insert(output_lines, output_line)
9116 end
9117 table.insert(output_lines, "}")
9118
9119 local output = table.concat(output_lines, "\n")
9120 local output_filename = options.debugExtensionsFileName
9121 local output_file = assert(io.open(output_filename, "w"),
9122 [[Could not open file]] .. output_filename .. [{" for writing}])
9123 assert(output_file:write(output))
9124 assert(output_file:close())
9125 end

```

Materialize [walkable\\_syntax](#) and merge it into [syntax](#) to produce the complete PEG grammar of markdown. Whenever a rule exists in both [walkable\\_syntax](#) and [syntax](#), the rule from [walkable\\_syntax](#) overrides the rule from [syntax](#).

```

9126 for lhs, rule in pairs(walkable_syntax) do
9127 syntax[lhs] = parsers.fail
9128 for _, rhs in ipairs(rule) do
9129 local pattern

```

Although the interface of the [reader->insert\\_pattern](#) method does not document this (see Section 2.1.2), we allow the [reader->insert\\_pattern](#) and [reader->update\\_rule](#) methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

9130 if type(rhs) == "string" then
9131 pattern = V(rhs)
9132 else
9133 pattern = rhs[1]
9134 if type(pattern) == "string" then
9135 pattern = V(pattern)
9136 end
9137 end
9138 syntax[lhs] = syntax[lhs] + pattern
9139 end

```

```
9140 end
```

Finalize the parser by reacting to options and by producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```
9141 if options.underscores then
9142 self.add_special_character("_")
9143 end
9144
9145 if not options.codeSpans then
9146 syntax.Code = parsers.fail
9147 else
9148 self.add_special_character("`")
9149 end
9150
9151 if not options.html then
9152 syntax.DisplayHtml = parsers.fail
9153 syntax.InlineHtml = parsers.fail
9154 syntax.HtmlEntity = parsers.fail
9155 else
9156 self.add_special_character("&")
9157 end
9158
9159 if options.preserveTabs then
9160 options.stripIndent = false
9161 end
9162
9163 if not options.smartEllipses then
9164 syntax.Smart = parsers.fail
9165 else
9166 self.add_special_character(".")
9167 end
9168
9169 if not options.relativeReferences then
9170 syntax.AutoLinkRelativeReference = parsers.fail
9171 end
9172
9173 if options.contentLevel == "inline" then
9174 syntax[1] = "Inlines"
9175 syntax.Inlines = V("InitializeState")
9176 * parsers.Inline^0
9177 * (parsers.spacing^0
9178 * parsers.eof / "")
9179 syntax.Space = parsers.Space + parsers.blankline / writer.space
9180 end
9181
9182 local blocks_nested_t = util.table_copy(syntax)
```

```

9183 blocks_nested_t.ExpectedJekyllData = parsers.fail
9184 parsers.blocks_nested = Ct(blocks_nested_t)
9185
9186 parsers.blocks = Ct(syntax)
9187
9188 local inlines_t = util.table_copy(syntax)
9189 inlines_t[1] = "Inlines"
9190 inlines_t.Inlines = V("InitializeState")
9191 * parsers.Inline^0
9192 * (parsers.spacing^0
9193 * parsers.eof / "")
9194 parsers.inlines = Ct(inlines_t)
9195
9196 local inlines_no_inline_note_t = util.table_copy(inlines_t)
9197 inlines_no_inline_note_t.InlineNote = parsers.fail
9198 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9199
9200 local inlines_no_html_t = util.table_copy(inlines_t)
9201 inlines_no_html_t.DisplayHtml = parsers.fail
9202 inlines_no_html_t.InlineHtml = parsers.fail
9203 inlines_no_html_t.HtmlEntity = parsers.fail
9204 parsers.inlines_no_html = Ct(inlines_no_html_t)
9205
9206 local inlines_nbsp_t = util.table_copy(inlines_t)
9207 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9208 inlines_nbsp_t.Space = parsers.NonbreakingSpace
9209 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9210
9211 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9212 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9213 inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9214 parsers.inlines_no_link_or_emphasis = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain T<sub>E</sub>X output and returns it..

```

9215 return function(input)

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

9216 input = input:gsub("\r\n?", "\n")
9217 if input:sub(-1) ~= "\n" then
9218 input = input .. "\n"
9219 end

```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```

9220 references = {}

```

```

9221 local opt_string = {}
9222 for k, _ in pairs(defaultOptions) do
9223 local v = options[k]
9224 if type(v) == "table" then
9225 for _, i in ipairs(v) do
9226 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9227 end
9228 elseif k ~= "cacheDir" then
9229 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9230 end
9231 end
9232 table.sort(opt_string)
9233 local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9234 local output
9235 local function convert(input)
9236 local document = self.parser_functions.parse_blocks(input)
9237 local output = util.rope_to_string(writer.document(document))

```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```

9238 local undosep_start, undosep_end
9239 local potential_secend_start, secend_start
9240 local potential_sep_start, sep_start
9241 while true do
9242 -- find a `writer->undosep`
9243 undosep_start, undosep_end = output:find(writer.undosep_text, 1, true)
9244 if undosep_start == nil then break end
9245 -- skip any preceding section ends
9246 secend_start = undosep_start
9247 while true do
9248 potential_secend_start = secend_start - #writer.secend_text
9249 if potential_secend_start < 1
9250 or output:sub(potential_secend_start, secend_start - 1) ~= writer.secend_text
9251 then break
9252 end
9253 secend_start = potential_secend_start
9254 end
9255 -- find an immediately preceding block element / paragraph separator
9256 sep_start = secend_start
9257 potential_sep_start = sep_start - #writer.interblocksep_text
9258 if potential_sep_start >= 1
9259 and output:sub(potential_sep_start, sep_start - 1) == writer.interblocksep_text
9260 then sep_start = potential_sep_start
9261 else
9262 potential_sep_start = sep_start - #writer.paragraphsep_text
9263 if potential_sep_start >= 1

```

```

9264 and output:sub(potential_sep_start, sep_start - 1) == writer.paragraph
9265 sep_start = potential_sep_start
9266 end
9267 end
9268 -- remove `writer->undosep` and immediately preceding block element / paragraph
9269 output = output:sub(1, sep_start - 1)
9270 .. output:sub(secend_start, undosep_start - 1)
9271 .. output:sub(undosep_end + 1)
9272 end
9273 return output
9274 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output via the `writer->pack` method.

```

9275 if options.eagerCache or options.finalizeCache then
9276 local name = util.cache(options.cacheDir, input, salt, convert,
9277 ".md" .. writer.suffix)
9278 output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

9279 else
9280 output = convert(input)
9281 end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

9282 if options.finalizeCache then
9283 local file, mode
9284 if options.frozenCacheCounter > 0 then
9285 mode = "a"
9286 else
9287 mode = "w"
9288 end
9289 file = assert(io.open(options.frozenCacheFileName, mode),
9290 [[Could not open file]] .. options.frozenCacheFileName
9291 .. [[for writing]])
9292 assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname]]
9293 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
9294 .. [[\\endcsname{]] .. output .. [[]]] .. "\\n"))
9295 assert(file:close())
9296 end
9297 return output
9298 end
9299 end
9300 return self
9301 end

```

### 3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```
9302 M.extensions = {}
```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
9303 M.extensions.bracketed_spans = function()
```

```
9304 return {
```

```
9305 name = "built-in bracketed_spans syntax extension",
```

```
9306 extend_writer = function(self)
```

Define `writer->span` as a function that will transform an input bracketed span `s` with attributes `attr` to the output format.

```
9307 function self.span(s, attr)
```

```
9308 if self.flatten_inlines then return s end
```

```
9309 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
```

```
9310 self.attributes(attr),
```

```
9311 s,
```

```
9312 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"
```

```
9313 end
```

```
9314 end, extend_reader = function(self)
```

```
9315 local parsers = self.parsers
```

```
9316 local writer = self.writer
```

```
9317
```

```
9318 local span_label = parsers.lbracket
```

```
9319 * (Cs((parsers.alphanumeric1
```

```
9320 + parsers.inticks
```

```
9321 + parsers.autolink
```

```
9322 + V("InlineHtml")
```

```
9323 + (parsers.backslash * parsers.backslash)
```

```
9324 + (parsers.backslash * (parsers.lbracket + parsers.rbracket
```

```
9325 + V("Space") + V("Endline")
```

```
9326 + (parsers.any
```

```
9327 - (parsers.newline + parsers.lbracket + parsers.rbracket
```

```
9328 + parsers.blankline2))))1)
```

```
9329 / self.parser_functions.parse_inlines)
```

```
9330 * parsers.rbracket
```

```
9331
```

```
9332 local Span = span_label
```

```
9333 * Ct(parsers.attributes)
```

```

9334 / writer.span
9335
9336 self.insert_pattern("Inline before LinkAndEmph",
9337 Span, "Span")
9338 end
9339 }
9340 end

```

### 3.1.7.2 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```

9341 M.extensions.citations = function(citation_nbsps)
9342 return {
9343 name = "built-in citations syntax extension",
9344 extend_writer = function(self)

```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is enabled, the citations should be rendered in-text, when applicable. The `cites` array contains tables with the following keys and values:

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.
- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.
- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```

9345 function self.citations(text_cites, cites)
9346 local buffer = {}
9347 if self.flatten_inlines then
9348 for _,cite in ipairs(cites) do
9349 if cite.prenote then
9350 table.insert(buffer, {cite.prenote, " "})
9351 end
9352 table.insert(buffer, cite.name)
9353 if cite.postnote then
9354 table.insert(buffer, {" ", cite.postnote})
9355 end
9356 end
9357 else

```



```

9358 table.insert(buffer, {"\\markdownRenderer", text_cites and "TextCite" or "C
9359 {"", #cites, "}"}))
9360 for _,cite in ipairs(cites) do
9361 table.insert(buffer, {cite.suppress_author and "-" or "+", "{",
9362 cite.prenote or "", "}{"", cite.postnote or "", "}{"", cite.name, "}"}))
9363 end
9364 end
9365 return buffer
9366 end
9367 end, extend_reader = function(self)
9368 local parsers = self.parsers
9369 local writer = self.writer
9370
9371 local citation_chars
9372 = parsers.alphanumeric
9373 + S("#$%&-+<>~/_")
9374
9375 local citation_name
9376 = Cs(parsers.dash^-1) * parsers.at
9377 * Cs(citation_chars
9378 * (((citation_chars + parsers.internal_punctuation
9379 - parsers.comma - parsers.semicolon)
9380 * -#((parsers.internal_punctuation - parsers.comma
9381 - parsers.semicolon)^0
9382 * -(citation_chars + parsers.internal_punctuation
9383 - parsers.comma - parsers.semicolon)))^0
9384 * citation_chars)^-1)
9385
9386 local citation_body_prenote
9387 = Cs((parsers.alphanumeric^1
9388 + parsers.bracketed
9389 + parsers.inticks
9390 + parsers.autolink
9391 + V("InlineHtml")
9392 + V("Space") + V("Endline")
9393 + (parsers.anyescaped
9394 - (parsers.newline + parsers.rbracket + parsers.blankline)
9395 - (parsers.spnl * parsers.dash^-1 * parsers.at))^1)
9396
9397 local citation_body_postnote
9398 = Cs((parsers.alphanumeric^1
9399 + parsers.bracketed
9400 + parsers.inticks
9401 + parsers.autolink
9402 + V("InlineHtml")
9403 + V("Space") + V("Endline")
9404 + (parsers.anyescaped

```

```

9405 - (parsers.newline + parsers.rbracket + parsers.semicolon
9406 + parsers.blankline^2))
9407 - (parsers.spnl * parsers.rbracket))^1)
9408
9409 local citation_body_chunk
9410 = (citation_body_prenote
9411 * parsers.spnlc_sep
9412 + Cc("")
9413 * parsers.spnlc
9414)
9415 * citation_name
9416 * (parsers.internal_punctuation - parsers.semicolon)^-
1
9417 * (parsers.spnlc
9418 * citation_body_postnote
9419 + Cc("")
9420 * parsers.spnlc
9421)
9422
9423 local citation_body
9424 = citation_body_chunk
9425 * (parsers.semicolon
9426 * parsers.spnlc
9427 * citation_body_chunk
9428)^0
9429
9430 local citation_headless_body_postnote
9431 = Cs((parsers.alphanumeric^1
9432 + parsers.bracketed
9433 + parsers.inticks
9434 + parsers.autolink
9435 + V("InlineHtml")
9436 + V("Space") + V("Endline")
9437 + (parsers.anyescaped
9438 - (parsers.newline + parsers.rbracket + parsers.at
9439 + parsers.semicolon + parsers.blankline^2))
9440 - (parsers.spnl * parsers.rbracket))^0)
9441
9442 local citation_headless_body
9443 = citation_headless_body_postnote
9444 * (parsers.semicolon
9445 * parsers.spnlc
9446 * citation_body_chunk
9447)^0
9448
9449 local citations
9450 = function(text_cites, raw_cites)

```

```

9451 local function normalize(str)
9452 if str == "" then
9453 str = nil
9454 else
9455 str = (citation_nbsps and
9456 self.parser_functions.parse_inlines_nbsp or
9457 self.parser_functions.parse_inlines)(str)
9458 end
9459 return str
9460 end
9461
9462 local cites = {}
9463 for i = 1,#raw_cites,4 do
9464 cites[#cites+1] = {
9465 prenote = normalize(raw_cites[i]),
9466 suppress_author = raw_cites[i+1] == "-",
9467 name = writer.identifier(raw_cites[i+2]),
9468 postnote = normalize(raw_cites[i+3]),
9469 }
9470 end
9471 return writer.citations(text_cites, cites)
9472 end
9473
9474 local TextCitations
9475 = Ct((parsers.spnlc
9476 * Cc("")
9477 * citation_name
9478 * ((parsers.spnlc
9479 * parsers.lbracket
9480 * citation_headless_body
9481 * parsers.rbracket) + Cc("")))^1)
9482 / function(raw_cites)
9483 return citations(true, raw_cites)
9484 end
9485
9486 local ParenthesizedCitations
9487 = Ct((parsers.spnlc
9488 * parsers.lbracket
9489 * citation_body
9490 * parsers.rbracket)^1)
9491 / function(raw_cites)
9492 return citations(false, raw_cites)
9493 end
9494
9495 local Citations = TextCitations + ParenthesizedCitations
9496
9497 self.insert_pattern("Inline before LinkAndEmph",

```

```

9498 Citations, "Citations")
9499
9500 self.add_special_character("@")
9501 self.add_special_character("-")
9502 end
9503 }
9504 end

```

### 3.1.7.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```

9505 M.extensions.content_blocks = function(language_map)

```

The `languages_json` table maps programming language filename extensions to fence infostrings. All `language_map` files located by the `kpathsea` library are loaded into a chain of tables. `languages_json` corresponds to the first table and is chained with the rest via Lua metatables.

```

9506 local languages_json = (function()
9507 local base, prev, curr
9508 for _, pathname in ipairs{kpse.lookup(language_map, { all=true })} do
9509 local file = io.open(pathname, "r")
9510 if not file then goto continue end
9511 local input = assert(file:read("*a"))
9512 assert(file:close())
9513 local json = input:gsub('[^\n]-:', '[%1]=')
9514 curr = load("_ENV = {}; return "..json")()
9515 if type(curr) == "table" then
9516 if base == nil then
9517 base = curr
9518 else
9519 setmetatable(prev, { __index = curr })
9520 end
9521 prev = curr
9522 end
9523 ::continue::
9524 end
9525 return base or {}
9526 end)()
9527
9528 return {
9529 name = "built-in content_blocks syntax extension",
9530 extend_writer = function(self)

```

Define `writer->contentblock` as a function that will transform an input iA Writer content block to the output format, where `src` corresponds to the URI prefix, `su` to

the URI extension, `type` to the type of the content block (`localfile` or `onlineimage`), and `tit` to the title of the content block.

```
9531 function self.contentblock(src,suf,type,tit)
9532 if not self.is_writing then return "" end
9533 src = src..".."..suf
9534 suf = suf:lower()
9535 if type == "onlineimage" then
9536 return {"\\markdownRendererContentBlockOnlineImage{"..suf.."}",
9537 {"",self.string(src),"}",
9538 {"",self.uri(src),"}",
9539 {"",self.string(tit or ""),}}
9540 elseif languages_json[suf] then
9541 return {"\\markdownRendererContentBlockCode{"..suf.."}",
9542 {"",self.string(languages_json[suf]),"}",
9543 {"",self.string(src),"}",
9544 {"",self.uri(src),"}",
9545 {"",self.string(tit or ""),}}
9546 else
9547 return {"\\markdownRendererContentBlock{"..suf.."}",
9548 {"",self.string(src),"}",
9549 {"",self.uri(src),"}",
9550 {"",self.string(tit or ""),}}
9551 end
9552 end
9553 end, extend_reader = function(self)
9554 local parsers = self.parsers
9555 local writer = self.writer
9556
9557 local contentblock_tail
9558 = parsers.optionaltitle
9559 * (parsers.newline + parsers.eof)
9560
9561 -- case insensitive online image suffix:
9562 local onlineimagesuffix
9563 = (function(...)
9564 local parser = nil
9565 for _, suffix in ipairs({...}) do
9566 local pattern=nil
9567 for i=1,#suffix do
9568 local char=suffix:sub(i,i)
9569 char = S(char:lower()..char:upper())
9570 if pattern == nil then
9571 pattern = char
9572 else
9573 pattern = pattern * char
9574 end
9575 end
9576 end
9577 end)
```

```

9576 if parser == nil then
9577 parser = pattern
9578 else
9579 parser = parser + pattern
9580 end
9581 end
9582 return parser
9583 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
9584
9585 -- online image url for iA Writer content blocks with mandatory suffix,
9586 -- allowing nested brackets:
9587 local onlineimageurl
9588 = (parsers.less
9589 * Cs((parsers.anyescaped
9590 - parsers.more
9591 - parsers.spacing
9592 - #(parsers.period
9593 * onlineimagesuffix
9594 * parsers.more
9595 * contentblock_tail))^0)
9596 * parsers.period
9597 * Cs(onlineimagesuffix)
9598 * parsers.more
9599 + (Cs((parsers.inparens
9600 + (parsers.anyescaped
9601 - parsers.spacing
9602 - parsers.rparent
9603 - #(parsers.period
9604 * onlineimagesuffix
9605 * contentblock_tail))))^0)
9606 * parsers.period
9607 * Cs(onlineimagesuffix))
9608) * Cc("onlineimage")
9609
9610 -- filename for iA Writer content blocks with mandatory suffix:
9611 local localfilepath
9612 = parsers.slash
9613 * Cs((parsers.anyescaped
9614 - parsers.tab
9615 - parsers.newline
9616 - #(parsers.period
9617 * parsers.alphanumeric^1
9618 * contentblock_tail))^1)
9619 * parsers.period
9620 * Cs(parsers.alphanumeric^1)
9621 * Cc("localfile")
9622

```

```

9623 local ContentBlock
9624 = parsers.check_trail_no_rem
9625 * (localfilepath + onlineimageurl)
9626 * contentblock_tail
9627 / writer.contentblock
9628
9629 self.insert_pattern("Block before Blockquote",
9630 ContentBlock, "ContentBlock")
9631 end
9632 }
9633 end

```

### 3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

9634 M.extensions.definition_lists = function(tight_lists)
9635 return {
9636 name = "built-in definition_lists syntax extension",
9637 extend_writer = function(self)

```

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```

9638 local function dliitem(term, defs)
9639 local retVal = {"\\markdownRendererDlItem{",term,"}"}
9640 for _, def in ipairs(defs) do
9641 retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
9642 "\\markdownRendererDlDefinitionEnd "}
9643 end
9644 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
9645 return retVal
9646 end
9647
9648 function self.definitionlist(items,tight)
9649 if not self.is_writing then return "" end
9650 local buffer = {}
9651 for _,item in ipairs(items) do
9652 buffer[#buffer + 1] = dliitem(item.term, item.definitions)
9653 end
9654 if tight and tight_lists then
9655 return {"\\markdownRendererDlBeginTight\n", buffer,
9656 "\n\\markdownRendererDlEndTight"}
9657 else
9658 return {"\\markdownRendererDlBegin\n", buffer,

```

```

9659 "\n\\markdownRendererDlEnd"}
9660 end
9661 end
9662 end, extend_reader = function(self)
9663 local parsers = self.parsers
9664 local writer = self.writer
9665
9666 local defstartchar = S("~:")
9667
9668 local defstart = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
9669 * (parsers.tab + parsers.space^-
3)
9670 + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
9671 * (parsers.tab + parsers.space^-
2)
9672 + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
9673 * (parsers.tab + parsers.space^-
1)
9674 + parsers.check_trail_length(3) * defstartchar * #parsers.spaci
9675
9676 local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_
9677
9678 local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option
9679
9680 local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
9681
9682 local indented_blocks = function(bl)
9683 return Cs(bl
9684 * (blank^1 * (parsers.check_minimal_indent / ""))
9685 * parsers.check_code_trail * -parsers.blankline * bl)^0
9686 * (blank^1 + parsers.eof))
9687 end
9688
9689 local function definition_list_item(term, defs, _)
9690 return { term = self.parser_functions.parse_inlines(term),
9691 definitions = defs }
9692 end
9693
9694 local DefinitionListItemLoose
9695 = C(parsers.line) * blank^0
9696 * Ct((parsers.check_minimal_indent * (defstart
9697 * indented_blocks(dlchunk)
9698 / self.parser_functions.parse_blocks_nested))^1)
9699 * Cc(false) / definition_list_item
9700
9701 local DefinitionListItemTight
9702 = C(parsers.line)

```



```

9703 * Ct((parsers.check_minimal_indent * (defstart * dlchunk
9704 / self.parser_functions.parse_blocks_nested))^1)
9705 * Cc(true) / definition_list_item
9706
9707 local DefinitionList
9708 = (Ct(DefinitionListItemLoose^1) * Cc(false)
9709 + Ct(DefinitionListItemTight^1)
9710 * (blank^0
9711 * -DefinitionListItemLoose * Cc(true))
9712) / writer.definitionlist
9713
9714 self.insert_pattern("Block after Heading",
9715 DefinitionList, "DefinitionList")
9716 end
9717 }
9718 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

9719 M.extensions.fancy_lists = function()
9720 return {
9721 name = "built-in fancy_lists syntax extension",
9722 extend_writer = function(self)
9723 local options = self.options
9724

```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
  - `Default` – default style,

- [OneParen](#) – parentheses, and
- [Period](#) – periods.

```

9725 function self.fancylist(items,tight,startnum,numstyle,numdelim)
9726 if not self.is_writing then return "" end
9727 local buffer = {}
9728 local num = startnum
9729 for _,item in ipairs(items) do
9730 if item ~= "" then
9731 buffer[#buffer + 1] = self.fancyitem(item,num)
9732 end
9733 if num ~= nil and item ~= "" then
9734 num = num + 1
9735 end
9736 end
9737 local contents = util.intersperse(buffer,"\n")
9738 if tight and options.tightLists then
9739 return {"\\markdownRendererFancyOlBeginTight{",
9740 numstyle,"}{",numdelim,"}",contents,
9741 "\\n\\markdownRendererFancyOlEndTight "}
9742 else
9743 return {"\\markdownRendererFancyOlBegin{",
9744 numstyle,"}{",numdelim,"}",contents,
9745 "\\n\\markdownRendererFancyOlEnd "}
9746 end
9747 end

```

Define [writer->fancyitem](#) as a function that will transform an input fancy ordered list item to the output format, where [s](#) is the text of the list item. If the optional parameter [num](#) is present, it is the number of the list item.

```

9748 function self.fancyitem(s,num)
9749 if num ~= nil then
9750 return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
9751 "\\markdownRendererFancyOlItemEnd "}
9752 else
9753 return {"\\markdownRendererFancyOlItem ",s,"\\markdownRendererFancyOlItemEnd "}
9754 end
9755 end
9756 end, extend_reader = function(self)
9757 local parsers = self.parsers
9758 local options = self.options
9759 local writer = self.writer
9760
9761 local function combine_markers_and_delims(markers, delims)
9762 local markers_table = {}
9763 for _,marker in ipairs(markers) do
9764 local start_marker

```

```

9765 local continuation_marker
9766 if type(marker) == "table" then
9767 start_marker = marker[1]
9768 continuation_marker = marker[2]
9769 else
9770 start_marker = marker
9771 continuation_marker = marker
9772 end
9773 for _,delim in ipairs(delims) do
9774 table.insert(markers_table, {start_marker, continuation_marker, delim})
9775 end
9776 end
9777 return markers_table
9778 end
9779
9780 local function join_table_with_func(func, markers_table)
9781 local pattern = func(table.unpack(markers_table[1]))
9782 for i = 2, #markers_table do
9783 pattern = pattern + func(table.unpack(markers_table[i]))
9784 end
9785 return pattern
9786 end
9787
9788 local lowercase_letter_marker = R("az")
9789 local uppercase_letter_marker = R("AZ")
9790
9791 local roman_marker = function(chars)
9792 local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
9793 local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
9794 return m^-3
9795 * (c*m + c*d + d^-1 * c^-3)
9796 * (x*c + x*l + l^-1 * x^-3)
9797 * (i*x + i*v + v^-1 * i^-3)
9798 end
9799
9800 local lowercase_roman_marker = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
9801 local uppercase_roman_marker = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
9802
9803 local lowercase_opening_roman_marker = P("i")
9804 local uppercase_opening_roman_marker = P("I")
9805
9806 local digit_marker = parsers.dig * parsers.dig^-8
9807
9808 local markers = {
9809 {lowercase_opening_roman_marker, lowercase_roman_marker},
9810 {uppercase_opening_roman_marker, uppercase_roman_marker},
9811 lowercase_letter_marker,

```

```

9812 uppercase_letter_marker,
9813 lowercase_roman_marker,
9814 uppercase_roman_marker,
9815 digit_marker
9816 }
9817
9818 local delims = {
9819 parsers.period,
9820 parsers.rparent
9821 }
9822
9823 local markers_table = combine_markers_and_delims(markers, delims)
9824
9825 local function enumerator(start_marker, _, delimiter_type, interrupting)
9826 local delimiter_range
9827 local allowed_end
9828 if interrupting then
9829 delimiter_range = P("1")
9830 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9831 else
9832 delimiter_range = start_marker
9833 allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
9834 end
9835
9836 return parsers.check_trail
9837 * Ct(C(delimiter_range) * C(delimiter_type))
9838 * allowed_end
9839 end
9840
9841 local starter = join_table_with_func(enumerator, markers_table)
9842
9843 local TightListItem = function(starter)
9844 return parsers.add_indent(starter, "li")
9845 * parsers.indented_content_tight
9846 end
9847
9848 local LooseListItem = function(starter)
9849 return parsers.add_indent(starter, "li")
9850 * parsers.indented_content_loose
9851 * remove_indent("li")
9852 end
9853
9854 local function roman2number(roman)
9855 local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
9856 local numeral = 0
9857
9858 local i = 1

```

```

9859 local len = string.len(roman)
9860 while i < len do
9861 local z1, z2 = romans[string.sub(roman, i, i)], romans[string.sub(roman,
9862 if z1 < z2 then
9863 numeral = numeral + (z2 - z1)
9864 i = i + 2
9865 else
9866 numeral = numeral + z1
9867 i = i + 1
9868 end
9869 end
9870 if i <= len then numeral = numeral + romans[string.sub(roman,i,i)] end
9871 return numeral
9872 end
9873
9874 local function sniffstyle(numstr, delimend)
9875 local numdelim
9876 if delimend == ")" then
9877 numdelim = "OneParen"
9878 elseif delimend == "." then
9879 numdelim = "Period"
9880 else
9881 numdelim = "Default"
9882 end
9883
9884 local num
9885 num = numstr:match("^([I])$")
9886 if num then
9887 return roman2number(num), "UpperRoman", numdelim
9888 end
9889 num = numstr:match("^([i])$")
9890 if num then
9891 return roman2number(string.upper(num)), "LowerRoman", numdelim
9892 end
9893 num = numstr:match("^([A-Z])$")
9894 if num then
9895 return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
9896 end
9897 num = numstr:match("^([a-z])$")
9898 if num then
9899 return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
9900 end
9901 num = numstr:match("^([IVXLCDM]+)")
9902 if num then
9903 return roman2number(num), "UpperRoman", numdelim
9904 end
9905 num = numstr:match("^([ivxlcdm]+)")

```

```

9906 if num then
9907 return roman2number(string.upper(num)), "LowerRoman", numdelim
9908 end
9909 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
9910 end
9911
9912 local function fancylist(items,tight,start)
9913 local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
9914 return writer.fancylist(items,tight,
9915 options.startNumber and startnum or 1,
9916 numstyle or "Decimal",
9917 numdelim or "Default")
9918 end
9919
9920 local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
9921 local enumerator_start = enumerator(start_marker, continuation_marker, delimi
9922 local enumerator_cont = enumerator(continuation_marker, continuation_marker,
9923 return Cg(enumerator_start, "listtype")
9924 * (Ct(TightListItem(Cb("listtype"))
9925 * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
9926 * Cc(true)
9927 * -#((parsers.conditionally_indented_blankline^0 / ""))
9928 * parsers.check_minimal_indent * enumerator_cont)
9929 + Ct(LooseListItem(Cb("listtype"))
9930 * ((parsers.conditionally_indented_blankline^0 / ""))
9931 * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_c
9932 * Cc(false)
9933) * Ct(Cb("listtype")) / fancylist
9934 end
9935
9936 local FancyList = join_table_with_func(FancyListOfType, markers_table)
9937
9938 local Endline = parsers.newline
9939 * (parsers.check_minimal_indent
9940 * -parsers.EndlineExceptions
9941 + parsers.check_optional_indent
9942 * -parsers.EndlineExceptions
9943 * -starter)
9944 * parsers.spacechar^0
9945 / writer.soft_line_break
9946
9947 self.update_rule("OrderedList", FancyList)
9948 self.update_rule("Endline", Endline)
9949 end
9950 }
9951 end

```

### 3.1.7.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```
9952 M.extensions.fenced_code = function(blank_before_code_fence,
9953 allow_attributes,
9954 allow_raw_blocks)
9955 return {
9956 name = "built-in fenced_code syntax extension",
9957 extend_writer = function(self)
9958 local options = self.options
9959
```

Define `writer->fencedCode` as a function that will transform an input fenced code block `s` with the infostring `i` and optional attributes `attr` to the output format.

```
9960 function self.fencedCode(s, i, attr)
9961 if not self.is_writing then return "" end
9962 s = s:gsub("\n$", "")
9963 local buf = {}
9964 if attr ~= nil then
9965 table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
9966 self.attributes(attr)})
9967 end
9968 local name = util.cache_verbatim(options.cacheDir, s)
9969 table.insert(buf, {"\\markdownRendererInputFencedCode{" ,
9970 name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
9971 if attr ~= nil then
9972 table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd")
9973 end
9974 return buf
9975 end
9976
```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```
9977 if allow_raw_blocks then
9978 function self.rawBlock(s, attr)
9979 if not self.is_writing then return "" end
9980 s = s:gsub("\n$", "")
9981 local name = util.cache_verbatim(options.cacheDir, s)
9982 return {"\\markdownRendererInputRawBlock{" ,
```

```

9983 name,"}{" , self.string(attr),"}"}
9984 end
9985 end
9986 end, extend_reader = function(self)
9987 local parsers = self.parsers
9988 local writer = self.writer
9989
9990 local function captures_geq_length(_,i,a,b)
9991 return #a >= #b and i
9992 end
9993
9994 local function strip_enclosing_whitespaces(str)
9995 return str:gsub("^%s*(.)%s*$", "%1")
9996 end
9997
9998 local tilde_infostring = Cs(Cs((V("HtmlEntity")
9999 + parsers.anyescaped
10000 - parsers.newline)^0)
10001 / strip_enclosing_whitespaces)
10002
10003 local backtick_infostring = Cs(Cs((V("HtmlEntity")
10004 + (-#(parsers.backslash * parsers.backtick) *
10005 - parsers.newline
10006 - parsers.backtick)^0)
10007 / strip_enclosing_whitespaces)
10008
10009 local fenceindent
10010
10011 local function has_trail(indent_table)
10012 return indent_table ~= nil and
10013 indent_table.trail ~= nil and
10014 next(indent_table.trail) ~= nil
10015 end
10016
10017 local function has_indents(indent_table)
10018 return indent_table ~= nil and
10019 indent_table.indents ~= nil and
10020 next(indent_table.indents) ~= nil
10021 end
10022
10023 local function get_last_indent_name(indent_table)
10024 if has_indents(indent_table) then
10025 return indent_table.indents[#indent_table.indents].name
10026 end
10027 end
10028
10029 local function count_fenced_start_indent(_, _, indent_table, trail)

```



```

10030 local last_indent_name = get_last_indent_name(indent_table)
10031 fenceindent = 0
10032 if last_indent_name ~= "li" then
10033 fenceindent = #trail
10034 end
10035 return true
10036 end
10037
10038 local fencehead = function(char, infostring)
10039 return Cmt(Cb("indent_info") * parsers.check_trail, count_fence
10040 * Cg(char^3, "fencelength")
10041 * parsers.optionalspace
10042 * infostring
10043 * (parsers.newline + parsers.eof)
10044 end
10045
10046 local fencetail = function(char)
10047 return parsers.check_trail_no_rem
10048 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10049 * parsers.optionalspace * (parsers.newline + parsers.eof)
10050 + parsers.eof
10051 end
10052
10053 local function process_fenced_line(s, i, indent_table, line_content, is_blank)
10054 local remainder = ""
10055 if has_trail(indent_table) then
10056 remainder = indent_table.trail.internal_remainder
10057 end
10058
10059 if is_blank and get_last_indent_name(indent_table) == "li" then
10060 remainder = ""
10061 end
10062
10063 local str = remainder .. line_content
10064 local index = 1
10065 local remaining = fenceindent
10066
10067 while true do
10068 local c = str:sub(index, index)
10069 if c == " " and remaining > 0 then
10070 remaining = remaining - 1
10071 index = index + 1
10072 elseif c == "\t" and remaining > 3 then
10073 remaining = remaining - 4
10074 index = index + 1
10075 else
10076 break

```

```

10077 end
10078 end
10079
10080 return true, str:sub(index)
10081 end
10082
10083 local fencedline = function(char)
10084 return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),
10085 end
10086
10087 local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true)
10088
10089 local TildeFencedCode
10090 = fencehead(parsers.tilde, tilde_infostring)
10091 * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10092 + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
10093 * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
10094
10095 local BacktickFencedCode
10096 = fencehead(parsers.backtick, backtick_infostring)
10097 * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10098 + (parsers.check_minimal_indent / "") * fencedline(parsers.backtick)
10099 * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10100
10101 local infostring_with_attributes
10102 = Ct(C((parsers.linechar
10103 - (parsers.optionalspace
10104 * parsers.attributes))^0)
10105 * parsers.optionalspace
10106 * Ct(parsers.attributes))
10107
10108 local FencedCode
10109 = ((TildeFencedCode + BacktickFencedCode)
10110 / function(infostring, code)
10111 local expanded_code = self.expandtabs(code)
10112
10113 if allow_raw_blocks then
10114 local raw_attr = lpeg.match(parsers.raw_attribute,
10115 infostring)
10116
10117 if raw_attr then
10118 return writer.rawBlock(expanded_code, raw_attr)
10119 end
10120 end
10121
10122 local attr = nil
10123 if allow_attributes then
10124 local match = lpeg.match(infostring_with_attributes,

```

```

10124 infostring)
10125 if match then
10126 infostring, attr = table.unpack(match)
10127 end
10128 end
10129 return writer.fencedCode(expanded_code, infostring, attr)
10130 end)
10131
10132 self.insert_pattern("Block after Verbatim",
10133 FencedCode, "FencedCode")
10134
10135 local fencestart
10136 if blank_before_code_fence then
10137 fencestart = parsers.fail
10138 else
10139 fencestart = fencehead(parsers.backtick, backtick_infostring)
10140 + fencehead(parsers.tilde, tilde_infostring)
10141 end
10142
10143 self.update_rule("EndlineExceptions", function(previous_pattern)
10144 if previous_pattern == nil then
10145 previous_pattern = parsers.EndlineExceptions
10146 end
10147 return previous_pattern + fencestart
10148 end)
10149
10150 self.add_special_character("`")
10151 self.add_special_character("~")
10152 end
10153 }
10154 end

```

### 3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```

10155 M.extensions.fenced_divs = function(blank_before_div_fence)
10156 return {
10157 name = "built-in fenced_divs syntax extension",
10158 extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```

10159 function self.div_begin(attributes)
10160 local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",

```

```

10161 self.attributes(attributes)}
10162 local end_output = {"\\markdownRendererFencedDivAttributeContextEnd "}
10163 return self.push_attributes("div", attributes, start_output, end_output)
10164 end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

10165 function self.div_end()
10166 return self.pop_attributes("div")
10167 end
10168 end, extend_reader = function(self)
10169 local parsers = self.parsers
10170 local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

10171 local fenced_div_infostring
10172 = C((parsers.linechar
10173 - (parsers.spacechar^1
10174 * parsers.colon^1))^1)
10175
10176 local fenced_div_begin = parsers.nonindentspace
10177 * parsers.colon^3
10178 * parsers.optionalspace
10179 * fenced_div_infostring
10180 * (parsers.spacechar^1
10181 * parsers.colon^1)^0
10182 * parsers.optionalspace
10183 * (parsers.newline + parsers.eof)
10184
10185 local fenced_div_end = parsers.nonindentspace
10186 * parsers.colon^3
10187 * parsers.optionalspace
10188 * (parsers.newline + parsers.eof)

```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```

10189 self.initialize_named_group("fenced_div_level", "0")
10190 self.initialize_named_group("fenced_div_num_opening_indents")
10191
10192 local function increment_div_level()
10193 local function push_indent_table(s, i, indent_table, -- luacheck: ignore s i
10194 fenced_div_num_opening_indents, fenced_div_l
10195 fenced_div_level = tonumber(fenced_div_level) + 1
10196 local num_opening_indents = 0

```

```

10197 if indent_table.indents ~= nil then
10198 num_opening_indents = #indent_table.indents
10199 end
10200 fenced_div_num_opening_indents[fenced_div_level] = num_opening_indents
10201 return true, fenced_div_num_opening_indents
10202 end
10203
10204 local function increment_level(s, i, fenced_div_level) -- luacheck: ignore s
10205 fenced_div_level = tonumber(fenced_div_level) + 1
10206 return true, tostring(fenced_div_level)
10207 end
10208
10209 return Cg(Cmt(Cb("indent_info")
10210 * Cb("fenced_div_num_opening_indents")
10211 * Cb("fenced_div_level"), push_indent_table)
10212 , "fenced_div_num_opening_indents")
10213 * Cg(Cmt(Cb("fenced_div_level"), increment_level)
10214 , "fenced_div_level")
10215 end
10216
10217 local function decrement_div_level()
10218 local function pop_indent_table(s, i, fenced_div_indent_table, fenced_div_level)
10219 fenced_div_level = tonumber(fenced_div_level)
10220 fenced_div_indent_table[fenced_div_level] = nil
10221 return true, tostring(fenced_div_level - 1)
10222 end
10223
10224 return Cg(Cmt(Cb("fenced_div_num_opening_indents")
10225 * Cb("fenced_div_level"), pop_indent_table)
10226 , "fenced_div_level")
10227 end
10228
10229
10230 local non_fenced_div_block = parsers.check_minimal_indent * V("Block")
10231 - parsers.check_minimal_indent_and_trail * fenced_div_block
10232
10233 local non_fenced_div_paragraph = parsers.check_minimal_indent * V("Paragraph")
10234 - parsers.check_minimal_indent_and_trail * fenced_div_block
10235
10236 local blank = parsers.minimally_indented_blank
10237
10238 local block_separated = parsers.block_sep_group(blank)
10239 * non_fenced_div_block
10240
10241 local loop_body_pair = parsers.create_loop_body_pair(block_separated,
10242 non_fenced_div_paragraph,
10243 parsers.block_sep_group(b

```

```

10244 parsers.par_sep_group(bla
10245
10246 local content_loop = (non_fenced_div_block
10247 * loop_body_pair.block^0
10248 + non_fenced_div_paragraph
10249 * block_separated
10250 * loop_body_pair.block^0
10251 + non_fenced_div_paragraph
10252 * loop_body_pair.par^0)
10253 * blank^0
10254
10255 local FencedDiv = fenced_div_begin
10256 / function (infostring)
10257 local attr = lpeg.match(Ct(parsers.attributes), infostring)
10258 if attr == nil then
10259 attr = {"." .. infostring}
10260 end
10261 return attr
10262 end
10263 / writer.div_begin
10264 * increment_div_level()
10265 * parsers.skipblanklines
10266 * Ct(content_loop)
10267 * parsers.minimally_indented_blank^0
10268 * parsers.check_minimal_indent_and_trail * fenced_div_end
10269 * decrement_div_level()
10270 * (Cc("") / writer.div_end)
10271
10272 self.insert_pattern("Block after Verbatim",
10273 FencedDiv, "FencedDiv")
10274
10275 self.add_special_character(":")
10276

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```

10277 local function is_inside_div()
10278 local function check_div_level(s, i, fenced_div_level) -- luacheck: ignore s
10279 fenced_div_level = tonumber(fenced_div_level)
10280 return fenced_div_level > 0
10281 end
10282
10283 return Cmt(Cb("fenced_div_level"), check_div_level)
10284 end
10285
10286 local function check_indent()

```

```

10287 local function compare_indent(s, i, indent_table, -- luacheck: ignore s i
10288 fenced_div_num_opening_indents, fenced_div_level)
10289 fenced_div_level = tonumber(fenced_div_level)
10290 local num_current_indents = (indent_table.current_line_indents ~= nil and
10291 #indent_table.current_line_indents) or 0
10292 local num_opening_indents = fenced_div_num_opening_indents[fenced_div_level]
10293 return num_current_indents == num_opening_indents
10294 end
10295
10296 return Cmt(Cb("indent_info")
10297 * Cb("fenced_div_num_opening_indents")
10298 * Cb("fenced_div_level"), compare_indent)
10299 end
10300
10301 local fencestart = is_inside_div()
10302 * fenced_div_end
10303 * check_indent()
10304
10305 if not blank_before_div_fence then
10306 self.update_rule("EndlineExceptions", function(previous_pattern)
10307 if previous_pattern == nil then
10308 previous_pattern = parsers.EndlineExceptions
10309 end
10310 return previous_pattern + fencestart
10311 end)
10312 end
10313 end
10314 }
10315 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

10316 M.extensions.header_attributes = function()
10317 return {
10318 name = "built-in header_attributes syntax extension",
10319 extend_writer = function()
10320 end, extend_reader = function(self)
10321 local parsers = self.parsers
10322 local writer = self.writer
10323
10324 local function strip_atx_end(s)
10325 return s:gsub("%s+##%s*$", "")
10326 end
10327
10328 local AtxHeading = Cg(parsers.heading_start, "level")

```

```

10329 * parsers.optionalspace
10330 * (C(((parsers.linechar
10331 - (parsers.attributes
10332 * parsers.optionalspace
10333 * parsers.newline))
10334 * (parsers.linechar
10335 - parsers.lbrace)^0)^1)
10336 / strip_atx_end
10337 / parsers.parse_heading_text)
10338 * Cg(Ct(parsers.newline
10339 + (parsers.attributes
10340 * parsers.optionalspace
10341 * parsers.newline)), "attributes")
10342 * Cb("level")
10343 * Cb("attributes")
10344 / writer.heading
10345
10346 local function strip_trailing_spaces(s)
10347 return s:gsub("%s*$", "")
10348 end
10349
10350 local heading_line = (parsers.linechar
10351 - (parsers.attributes
10352 * parsers.optionalspace
10353 * parsers.newline))^1
10354 - parsers.thematic_break_lines
10355
10356 local heading_text = heading_line
10357 * ((V("Endline") / "\n") * (heading_line - parsers.heading_
10358 * parsers.newline^-1)
10359
10360 local SettextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
10361 * #(heading_text
10362 * (parsers.attributes
10363 * parsers.optionalspace
10364 * parsers.newline)^-1
10365 * parsers.check_minimal_indent * parsers.check_trail *
10366 * Cs(heading_text) / strip_trailing_spaces
10367 / parsers.parse_heading_text
10368 * Cg(Ct((parsers.attributes
10369 * parsers.optionalspace
10370 * parsers.newline)^-1), "attributes")
10371 * parsers.check_minimal_indent_and_trail * parsers.heading_
10372 * Cb("attributes")
10373 * parsers.newline
10374 * parsers.unfreeze_trail
10375 / writer.heading

```



```

10376
10377 local Heading = AtxHeading + SetextHeading
10378 self.update_rule("Heading", Heading)
10379 end
10380 }
10381 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

10382 M.extensions.inline_code_attributes = function()
10383 return {
10384 name = "built-in inline_code_attributes syntax extension",
10385 extend_writer = function()
10386 end, extend_reader = function(self)
10387 local writer = self.writer
10388
10389 local CodeWithAttributes = parsers.inticks
10390 * Ct(parsers.attributes)
10391 / writer.code
10392
10393 self.insert_pattern("Inline before Code",
10394 CodeWithAttributes,
10395 "CodeWithAttributes")
10396 end
10397 }
10398 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

10399 M.extensions.line_blocks = function()
10400 return {
10401 name = "built-in line_blocks syntax extension",
10402 extend_writer = function(self)

```

Define `writer->lineblock` as a function that will transform a line block consisted of `lines` to the output format, with all but the last newline rendered as a line break.

```

10403 function self.lineblock(lines)
10404 if not self.is_writing then return "" end
10405 local buffer = {}
10406 for i = 1, #lines - 1 do
10407 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
10408 end
10409 buffer[#buffer + 1] = lines[#lines]
10410

```

```

10411 return {"\\markdownRendererLineBlockBegin\n"
10412 ,buffer,
10413 "\\n\\markdownRendererLineBlockEnd "}
10414 end
10415 end, extend_reader = function(self)
10416 local parsers = self.parsers
10417 local writer = self.writer
10418
10419 local LineBlock = Ct(
10420 (Cs(
10421 ((parsers.pipe * parsers.space)/""
10422 * ((parsers.space)/entities.char_entity("nbsp"))^0
10423 * parsers.linechar^0 * (parsers.newline/"")
10424 * (-parsers.pipe
10425 * (parsers.space^1/" ")
10426 * parsers.linechar^1
10427 * (parsers.newline/"")
10428)^0
10429 * (parsers.blankline/"")^0
10430) / self.parser_functions.parse_inlines)^1) / writer.lineblock
10431
10432 self.insert_pattern("Block after Blockquote",
10433 LineBlock, "LineBlock")
10434 end
10435 }
10436 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

10437 M.extensions.mark = function()
10438 return {
10439 name = "built-in mark syntax extension",
10440 extend_writer = function(self)

```

Define `writer->mark` as a function that will transform an input marked text `s` to the output format.

```

10441 function self.mark(s)
10442 if self.flatten_inlines then return s end
10443 return {"\\markdownRendererMark{" , s, "}" }
10444 end
10445 end, extend_reader = function(self)
10446 local parsers = self.parsers
10447 local writer = self.writer
10448
10449 local doubleequals = P("==")
10450
10451 local Mark = parsers.between(V("Inline"), doubleequals, doubleequals)

```

```

10452 / function (inlines) return writer.mark(inlines) end
10453
10454 self.add_special_character("=")
10455 self.insert_pattern("Inline before LinkAndEmph",
10456 Mark, "Mark")
10457 end
10458 }
10459 end

```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

10460 M.extensions.link_attributes = function()
10461 return {
10462 name = "built-in link_attributes syntax extension",
10463 extend_writer = function()
10464 end, extend_reader = function(self)
10465 local parsers = self.parsers
10466 local options = self.options
10467

```

The following patterns define link reference definitions with attributes.

```

10468 local define_reference_parser = (parsers.check_trail / "") * parsers.link_label
10469 * parsers.spnlc * parsers.url
10470 * (parsers.spnlc_sep * parsers.title * (parsers.
10471 * parsers.only_blank
10472 + parsers.spnlc_sep * parsers.title * parsers.c
10473 + Cc("")) * (parsers.spnlc * Ct(parsers.attribut
10474 + Cc("")) * parsers.only_blank)
10475
10476 local ReferenceWithAttributes = define_reference_parser
10477 / self.register_link
10478
10479 self.update_rule("Reference", ReferenceWithAttributes)
10480

```

The following patterns define direct and indirect links with attributes.

```

10481
10482 local LinkWithAttributesAndEmph = Ct(parsers.link_and_emph_table * Cg(Cc(true),
10483 / self.defer_link_and_emphasis_processing
10484
10485 self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
10486

```

The following patterns define autolinks with attributes.

```

10487 local AutoLinkUrlWithAttributes
10488 = parsers.auto_link_url

```

```

10489 * Ct(parsers.attributes)
10490 / self.auto_link_url
10491
10492 self.insert_pattern("Inline before AutoLinkUrl",
10493 AutoLinkUrlWithAttributes,
10494 "AutoLinkUrlWithAttributes")
10495
10496 local AutoLinkEmailWithAttributes
10497 = parsers.auto_link_email
10498 * Ct(parsers.attributes)
10499 / self.auto_link_email
10500
10501 self.insert_pattern("Inline before AutoLinkEmail",
10502 AutoLinkEmailWithAttributes,
10503 "AutoLinkEmailWithAttributes")
10504
10505 if options.relativeReferences then
10506
10507 local AutoLinkRelativeReferenceWithAttributes
10508 = parsers.auto_link_relative_reference
10509 * Ct(parsers.attributes)
10510 / self.auto_link_url
10511
10512 self.insert_pattern(
10513 "Inline before AutoLinkRelativeReference",
10514 AutoLinkRelativeReferenceWithAttributes,
10515 "AutoLinkRelativeReferenceWithAttributes")
10516
10517 end
10518
10519 end
10520 }
10521 end

```

### 3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```

10522 M.extensions.notes = function(notes, inline_notes)
10523 assert(notes or inline_notes)
10524 return {
10525 name = "built-in notes syntax extension",
10526 extend_writer = function(self)

```

Define `writer->note` as a function that will transform an input note `s` to the output format.

```
10527 function self.note(s)
10528 if self.flatten_inlines then return "" end
10529 return {"\\markdownRendererNote{",s,""}
10530 end
10531 end, extend_reader = function(self)
10532 local parsers = self.parsers
10533 local writer = self.writer
10534
10535 if inline_notes then
10536 local InlineNote
10537 = parsers.circumflex
10538 * (parsers.link_label / self.parser_functions.parse_inlines_no_in
10539 / writer.note
10540
10541 self.insert_pattern("Inline after LinkAndEmph",
10542 InlineNote, "InlineNote")
10543 end
10544 if notes then
10545 local function strip_first_char(s)
10546 return s:sub(2)
10547 end
10548
10549 local RawNoteRef
10550 = #(parsers.lbracket * parsers.circumflex)
10551 * parsers.link_label / strip_first_char
10552
10553 local rawnotes = {}
10554
10555 -- like indirect_link
10556 local function lookup_note(ref)
10557 return writer.defer_call(function()
10558 local found = rawnotes[self.normalize_tag(ref)]
10559 if found then
10560 return writer.note(
10561 self.parser_functions.parse_blocks_nested(found))
10562 else
10563 return {"[",
10564 self.parser_functions.parse_inlines("^" .. ref), "]" }
10565 end
10566 end)
10567 end
10568
10569 local function register_note(ref,rawnote)
10570 local normalized_tag = self.normalize_tag(ref)
10571 if rawnotes[normalized_tag] == nil then
```

```

10572 rawnotes[normalized_tag] = rawnote
10573 end
10574 return ""
10575 end
10576
10577 local NoteRef = RawNoteRef / lookup_note
10578
10579 local optionally_indented_line = parsers.check_optional_indent_and_any_trail
10580
10581 local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt
10582
10583 local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)
10584
10585 local indented_blocks = function(bl)
10586 return Cs(bl
10587 * (blank^1 * (parsers.check_optional_indent / ""))
10588 * parsers.check_code_trail * -parsers.blankline * bl)^0)
10589 end
10590
10591 local NoteBlock
10592 = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10593 * parsers.spnlc * indented_blocks(chunk)
10594 / register_note
10595
10596 local Reference = NoteBlock + parsers.Reference
10597
10598 self.update_rule("Reference", Reference)
10599 self.insert_pattern("Inline before LinkAndEmph",
10600 NoteRef, "NoteRef")
10601 end
10602
10603 self.add_special_character("^")
10604 end
10605 }
10606 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```

10607 M.extensions.pipe_tables = function(table_captions, table_attributes)
10608
10609 local function make_pipe_table_rectangular(rows)

```

```

10610 local num_columns = #rows[2]
10611 local rectangular_rows = {}
10612 for i = 1, #rows do
10613 local row = rows[i]
10614 local rectangular_row = {}
10615 for j = 1, num_columns do
10616 rectangular_row[j] = row[j] or ""
10617 end
10618 table.insert(rectangular_rows, rectangular_row)
10619 end
10620 return rectangular_rows
10621 end
10622
10623 local function pipe_table_row(allow_empty_first_column
10624 , nonempty_column
10625 , column_separator
10626 , column)
10627 local row_beginning
10628 if allow_empty_first_column then
10629 row_beginning = -- empty first column
10630 #(parsers.spacechar^4
10631 * column_separator)
10632 * parsers.optionalspace
10633 * column
10634 * parsers.optionalspace
10635 -- non-empty first column
10636 + parsers.nonindentspace
10637 * nonempty_column^-1
10638 * parsers.optionalspace
10639 else
10640 row_beginning = parsers.nonindentspace
10641 * nonempty_column^-1
10642 * parsers.optionalspace
10643 end
10644
10645 return Ct(row_beginning
10646 * (-- single column with no leading pipes
10647 #(column_separator
10648 * parsers.optionalspace
10649 * parsers.newline)
10650 * column_separator
10651 * parsers.optionalspace
10652 -- single column with leading pipes or
10653 -- more than a single column
10654 + (column_separator
10655 * parsers.optionalspace
10656 * column

```

```

10657 * parsers.optionalspace)^1
10658 * (column_separator
10659 * parsers.optionalspace)^-1))
10660 end
10661
10662 return {
10663 name = "built-in pipe_tables syntax extension",
10664 extend_writer = function(self)

```

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```

10665 function self.table(rows, caption, attributes)
10666 if not self.is_writing then return "" end
10667 local buffer = {}
10668 if attributes ~= nil then
10669 table.insert(buffer,
10670 "\\markdownRendererTableAttributeContextBegin\n")
10671 table.insert(buffer, self.attributes(attributes))
10672 end
10673 table.insert(buffer,
10674 {"\\markdownRendererTable{",
10675 caption or "", "}{"", #rows - 1, "}{"",
10676 #rows[1], "}"}))
10677 local temp = rows[2] -- put alignments on the first row
10678 rows[2] = rows[1]
10679 rows[1] = temp
10680 for i, row in ipairs(rows) do
10681 table.insert(buffer, "{")
10682 for _, column in ipairs(row) do
10683 if i > 1 then -- do not use braces for alignments
10684 table.insert(buffer, "{")
10685 end
10686 table.insert(buffer, column)
10687 if i > 1 then
10688 table.insert(buffer, "}")
10689 end
10690 end
10691 table.insert(buffer, "}")
10692 end
10693 if attributes ~= nil then
10694 table.insert(buffer,
10695 "\\markdownRendererTableAttributeContextEnd{")
10696 end
10697 return buffer
10698 end
10699 end, extend_reader = function(self)

```



```

10700 local parsers = self.parsers
10701 local writer = self.writer
10702
10703 local table_hline_separator = parsers.pipe + parsers.plus
10704
10705 local table_hline_column = (parsers.dash
10706 - #(parsers.dash
10707 * (parsers.spacechar
10708 + table_hline_separator
10709 + parsers.newline)))^1
10710 * (parsers.colon * Cc("r")
10711 + parsers.dash * Cc("d"))
10712 + parsers.colon
10713 * (parsers.dash
10714 - #(parsers.dash
10715 * (parsers.spacechar
10716 + table_hline_separator
10717 + parsers.newline)))^1
10718 * (parsers.colon * Cc("c")
10719 + parsers.dash * Cc("l"))
10720
10721 local table_hline = pipe_table_row(false
10722 , table_hline_column
10723 , table_hline_separator
10724 , table_hline_column)
10725
10726 local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
10727 * parsers.optionalspace * parsers.newline)^0
10728 * parsers.check_minimal_indent_and_trail
10729 * (P("Table")^-1 * parsers.colon)
10730 * parsers.optionalspace
10731
10732 local function strip_trailing_spaces(s)
10733 return s:gsub("%s*$", "")
10734 end
10735
10736 local table_row = pipe_table_row(true
10737 , (C((parsers.linechar - parsers.pipe)^1)
10738 / strip_trailing_spaces
10739 / self.parser_functions.parse_inlines)
10740 , parsers.pipe
10741 , (C((parsers.linechar - parsers.pipe)^0)
10742 / strip_trailing_spaces
10743 / self.parser_functions.parse_inlines))
10744
10745 local table_caption
10746 if table_captions then

```

```

10747 table_caption = #table_caption_beginning
10748 * table_caption_beginning
10749 if table_attributes then
10750 table_caption = table_caption
10751 * (C((((parsers.linechar
10752 - (parsers.attributes
10753 * parsers.optionalspace
10754 * parsers.newline
10755 * -(parsers.optionalspace
10756 * parsers.linechar)))
10757 + (parsers.newline
10758 * #(parsers.optionalspace
10759 * parsers.linechar)
10760 * C(parsers.optionalspace) / writer.space))
10761 * (parsers.linechar
10762 - parsers.lbrace)^0^1)
10763 / self.parser_functions.parse_inlines)
10764 * (parsers.newline
10765 + (Ct(parsers.attributes)
10766 * parsers.optionalspace
10767 * parsers.newline))
10768 else
10769 table_caption = table_caption
10770 * C((parsers.linechar
10771 + (parsers.newline
10772 * #(parsers.optionalspace
10773 * parsers.linechar)
10774 * C(parsers.optionalspace) / writer.space))^1)
10775 / self.parser_functions.parse_inlines
10776 * parsers.newline
10777 end
10778 else
10779 table_caption = parsers.fail
10780 end
10781
10782 local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_indent
10783 * table_hline * parsers.newline
10784 * ((parsers.check_minimal_indent / {}) * table_row * parsers.
10785 / make_pipe_table_rectangular
10786 * table_caption^-1
10787 / writer.table
10788
10789 self.insert_pattern("Block after Blockquote",
10790 PipeTable, "PipeTable")
10791 end
10792 }
10793 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
10794 M.extensions.raw_inline = function()
10795 return {
10796 name = "built-in raw_inline syntax extension",
10797 extend_writer = function(self)
10798 local options = self.options
10799
```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```
10800 function self.rawInline(s, attr)
10801 if not self.is_writing then return "" end
10802 if self.flatten_inlines then return s end
10803 local name = util.cache_verbatim(options.cacheDir, s)
10804 return {"\\markdownRendererInputRawInline{" ,
10805 name,"}{" , self.string(attr),"}" }
10806 end
10807 end, extend_reader = function(self)
10808 local writer = self.writer
10809
10810 local RawInline = parsers.inticks
10811 * parsers.raw_attribute
10812 / writer.rawInline
10813
10814 self.insert_pattern("Inline before Code",
10815 RawInline, "RawInline")
10816 end
10817 }
10818 end
```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
10819 M.extensions.strike_through = function()
10820 return {
10821 name = "built-in strike_through syntax extension",
10822 extend_writer = function(self)
```

Define `writer->strike_through` as a function that will transform a strike-through span `s` of input text to the output format.

```
10823 function self.strike_through(s)
10824 if self.flatten_inlines then return s end
10825 return {"\\markdownRendererStrikeThrough{" ,s,"}" }
10826 end
```

```

10827 end, extend_reader = function(self)
10828 local parsers = self.parsers
10829 local writer = self.writer
10830
10831 local StrikeThrough = (
10832 parsers.between(parsers.Inline, parsers.doubletildes,
10833 parsers.doubletildes)
10834) / writer.strike_through
10835
10836 self.insert_pattern("Inline after LinkAndEmph",
10837 StrikeThrough, "StrikeThrough")
10838
10839 self.add_special_character("~")
10840 end
10841 }
10842 end

```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

10843 M.extensions.subscripts = function()
10844 return {
10845 name = "built-in subscripts syntax extension",
10846 extend_writer = function(self)

```

Define `writer->subscript` as a function that will transform a subscript span `s` of input text to the output format.

```

10847 function self.subscript(s)
10848 if self.flatten_inlines then return s end
10849 return {"\\markdownRendererSubscript{" ,s, "}"}
10850 end
10851 end, extend_reader = function(self)
10852 local parsers = self.parsers
10853 local writer = self.writer
10854
10855 local Subscript = (
10856 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
10857) / writer.subscript
10858
10859 self.insert_pattern("Inline after LinkAndEmph",
10860 Subscript, "Subscript")
10861
10862 self.add_special_character("~")
10863 end
10864 }
10865 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```
10866 M.extensions.superscripts = function()
10867 return {
10868 name = "built-in superscripts syntax extension",
10869 extend_writer = function(self)
```

Define `writer->superscript` as a function that will transform a superscript span `s` of input text to the output format.

```
10870 function self.superscript(s)
10871 if self.flatten_inlines then return s end
10872 return {"\\markdownRendererSuperscript{" ,s,"}"}
10873 end
10874 end, extend_reader = function(self)
10875 local parsers = self.parsers
10876 local writer = self.writer
10877
10878 local Superscript = (
10879 parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
10880) / writer.superscript
10881
10882 self.insert_pattern("Inline after LinkAndEmph",
10883 Superscript, "Superscript")
10884
10885 self.add_special_character("^")
10886 end
10887 }
10888 end
```

### 3.1.7.19 TeX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
10889 M.extensions.tex_math = function(tex_math_dollars,
10890 tex_math_single_backslash,
10891 tex_math_double_backslash)
10892 return {
10893 name = "built-in tex_math syntax extension",
10894 extend_writer = function(self)
```

Define `writer->display_math` as a function that will transform a math span `s` of input text to the output format.

```
10895 function self.display_math(s)
10896 if self.flatten_inlines then return s end
10897 return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
10898 end
```

Define `writer->inline_math` as a function that will transform a math span `s` of input text to the output format.

```

10899 function self.inline_math(s)
10900 if self.flatten_inlines then return s end
10901 return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
10902 end
10903 end, extend_reader = function(self)
10904 local parsers = self.parsers
10905 local writer = self.writer
10906
10907 local function between(p, starter, ender)
10908 return (starter * Cs(p * (p - ender)^0) * ender)
10909 end
10910
10911 local function strip_preceding_whitespaces(str)
10912 return str:gsub("^%s*(.-)$", "%1")
10913 end
10914
10915 local allowed_before_closing = B(parsers.backslash * parsers.any
10916 + parsers.any * (parsers.any - parsers.backslash)
10917
10918 local allowed_before_closing_no_space = B(parsers.backslash * parsers.any
10919 + parsers.any * (parsers.nonspacechar
10920

```

The following patterns implement the Pandoc dollar math syntax extension.

```

10921 local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
10922 + parsers.backslash^-1
10923 * parsers.linechar)
10924 - parsers.blankline^2
10925 - parsers.dollar
10926
10927 local inline_math_opening_dollars = parsers.dollar
10928 * #(parsers.nonspacechar)
10929
10930 local inline_math_closing_dollars = allowed_before_closing_no_space
10931 * parsers.dollar
10932 * -#(parsers.digit)
10933
10934 local inline_math_dollars = between(Cs(dollar_math_content),
10935 inline_math_opening_dollars,
10936 inline_math_closing_dollars)
10937
10938 local display_math_opening_dollars = parsers.dollar
10939 * parsers.dollar
10940
10941 local display_math_closing_dollars = parsers.dollar

```

```

10942 * parsers.dollar
10943
10944 local display_math_dollars = between(Cs(dollar_math_content),
10945 display_math_opening_dollars,
10946 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

10947 local backslash_math_content = (parsers.newline * (parsers.check_optional_inde
10948 + parsers.linechar)
10949 - parsers.blankline^2)

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

10950 local inline_math_opening_double = parsers.backslash
10951 * parsers.backslash
10952 * parsers.lparent
10953
10954 local inline_math_closing_double = allowed_before_closing
10955 * parsers.spacechar^0
10956 * parsers.backslash
10957 * parsers.backslash
10958 * parsers.rparent
10959
10960 local inline_math_double = between(Cs(backslash_math_content),
10961 inline_math_opening_double,
10962 inline_math_closing_double)
10963 / strip_preceding_whitespaces
10964
10965 local display_math_opening_double = parsers.backslash
10966 * parsers.backslash
10967 * parsers.lbracket
10968
10969 local display_math_closing_double = allowed_before_closing
10970 * parsers.spacechar^0
10971 * parsers.backslash
10972 * parsers.backslash
10973 * parsers.rbracket
10974
10975 local display_math_double = between(Cs(backslash_math_content),
10976 display_math_opening_double,
10977 display_math_closing_double)
10978 / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

10979 local inline_math_opening_single = parsers.backslash
10980 * parsers.lparent
10981

```

```

10982 local inline_math_closing_single = allowed_before_closing
10983 * parsers.spacechar^0
10984 * parsers.backslash
10985 * parsers.rparent
10986
10987 local inline_math_single = between(Cs(backslash_math_content),
10988 inline_math_opening_single,
10989 inline_math_closing_single)
10990 / strip_preceding_whitespaces
10991
10992 local display_math_opening_single = parsers.backslash
10993 * parsers.lbracket
10994
10995 local display_math_closing_single = allowed_before_closing
10996 * parsers.spacechar^0
10997 * parsers.backslash
10998 * parsers.rbracket
10999
11000 local display_math_single = between(Cs(backslash_math_content),
11001 display_math_opening_single,
11002 display_math_closing_single)
11003 / strip_preceding_whitespaces
11004
11005 local display_math = parsers.fail
11006
11007 local inline_math = parsers.fail
11008
11009 if tex_math_dollars then
11010 display_math = display_math + display_math_dollars
11011 inline_math = inline_math + inline_math_dollars
11012 end
11013
11014 if tex_math_double_backslash then
11015 display_math = display_math + display_math_double
11016 inline_math = inline_math + inline_math_double
11017 end
11018
11019 if tex_math_single_backslash then
11020 display_math = display_math + display_math_single
11021 inline_math = inline_math + inline_math_single
11022 end
11023
11024 local TexMath = display_math / writer.display_math
11025 + inline_math / writer.inline_math
11026
11027 self.insert_pattern("Inline after LinkAndEmph",
11028 TexMath, "TexMath")

```



```

11029
11030 if tex_math_dollars then
11031 self.add_special_character("$")
11032 end
11033
11034 if tex_math_single_backslash or tex_math_double_backslash then
11035 self.add_special_character("\\")
11036 self.add_special_character("[")
11037 self.add_special_character("]")
11038 self.add_special_character("(")
11039 self.add_special_character("(")
11040 end
11041 end
11042 }
11043 end

```

### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

```

11044 M.extensions.jekyll_data = function(expect_jekyll_data)
11045 return {
11046 name = "built-in jekyll_data syntax extension",
11047 extend_writer = function(self)

```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t`.

```

11048 function self.jekyllData(d, t, p)
11049 if not self.is_writing then return "" end
11050
11051 local buf = {}
11052
11053 local keys = {}
11054 for k, _ in pairs(d) do
11055 table.insert(keys, k)
11056 end
11057 table.sort(keys)
11058
11059 if not p then
11060 table.insert(buf, "\\markdownRendererJekyllDataBegin")
11061 end
11062

```

```

11063 if #d > 0 then
11064 table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
11065 table.insert(buf, self.identified(p or "null"))
11066 table.insert(buf, "}{"")
11067 table.insert(buf, #keys)
11068 table.insert(buf, "}")
11069 else
11070 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
11071 table.insert(buf, self.identified(p or "null"))
11072 table.insert(buf, "}{"")
11073 table.insert(buf, #keys)
11074 table.insert(buf, "}")
11075 end
11076
11077 for _, k in ipairs(keys) do
11078 local v = d[k]
11079 local typ = type(v)
11080 k = tostring(k or "null")
11081 if typ == "table" and next(v) ~= nil then
11082 table.insert(
11083 buf,
11084 self.jekyllData(v, t, k)
11085)
11086 else
11087 k = self.identified(k)
11088 v = tostring(v)
11089 if typ == "boolean" then
11090 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11091 table.insert(buf, k)
11092 table.insert(buf, "}{"")
11093 table.insert(buf, v)
11094 table.insert(buf, "}")
11095 elseif typ == "number" then
11096 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11097 table.insert(buf, k)
11098 table.insert(buf, "}{"")
11099 table.insert(buf, v)
11100 table.insert(buf, "}")
11101 elseif typ == "string" then
11102 table.insert(buf, "\\markdownRendererJekyllDataString{")
11103 table.insert(buf, k)
11104 table.insert(buf, "}{"")
11105 table.insert(buf, t(v))
11106 table.insert(buf, "}")
11107 elseif typ == "table" then
11108 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11109 table.insert(buf, k)

```

```

11110 table.insert(buf, "}")
11111 else
11112 error(format("Unexpected type %s for value of " ..
11113 "YAML key %s", typ, k))
11114 end
11115 end
11116 end
11117
11118 if #d > 0 then
11119 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
11120 else
11121 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
11122 end
11123
11124 if not p then
11125 table.insert(buf, "\\markdownRendererJekyllDataEnd")
11126 end
11127
11128 return buf
11129 end
11130 end, extend_reader = function(self)
11131 local parsers = self.parsers
11132 local writer = self.writer
11133
11134 local JekyllData
11135 = Cmt(C((parsers.line - P("----") - P("..."))^0)
11136 , function(s, i, text) -- luacheck: ignore s i
11137 local data
11138 local ran_ok, _ = pcall(function()
11139 -- TODO: Replace with `require("tinyyaml")` in TeX Live
11140 local tinyyaml = require("markdown-tinyyaml")
11141 data = tinyyaml.parse(text, {timestamps=false})
11142 end)
11143 if ran_ok and data ~= nil then
11144 return true, writer.jekyllData(data, function(s)
11145 return self.parser_functions.parse_blocks_nested(s)
11146 end, nil)
11147 else
11148 return false
11149 end
11150 end
11151)
11152
11153 local UnexpectedJekyllData
11154 = P("----")
11155 * parsers.blankline / 0
11156 * #(-parsers.blankline) -- if followed by blank, it's thematic b

```

```

11157 * JekyllData
11158 * (P("----") + P("..."))
11159
11160 local ExpectedJekyllData
11161 = (P("----")
11162 * parsers.blankline / 0
11163 * #(-parsers.blankline) -- if followed by blank, it's thematic
11164)^-1
11165 * JekyllData
11166 * (P("----") + P("..."))^-1
11167
11168 self.insert_pattern("Block before Blockquote",
11169 UnexpectedJekyllData, "UnexpectedJekyllData")
11170 if expect_jekyll_data then
11171 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
11172 end
11173 end
11174 }
11175 end

```

### 3.1.8 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```
11176 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```

11177 options = options or {}
11178 setmetatable(options, { __index = function (_, key)
11179 return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

11180 if options.singletonCache and singletonCache.convert then
11181 for k, v in pairs(defaultOptions) do
11182 if type(v) == "table" then
11183 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
11184 if singletonCache.options[k][i] ~= options[k][i] then
11185 goto miss
11186 end
11187 end
11188 elseif singletonCache.options[k] ~= options[k] then
11189 goto miss
11190 end
11191 end
11192 return singletonCache.convert
11193 end

```

```

11194 ::miss::
 Apply built-in syntax extensions based on options.
11195 local extensions = {}
11196
11197 if options.bracketedSpans then
11198 local bracketed_spans_extension = M.extensions.bracketed_spans()
11199 table.insert(extensions, bracketed_spans_extension)
11200 end
11201
11202 if options.contentBlocks then
11203 local content_blocks_extension = M.extensions.content_blocks(
11204 options.contentBlocksLanguageMap)
11205 table.insert(extensions, content_blocks_extension)
11206 end
11207
11208 if options.definitionLists then
11209 local definition_lists_extension = M.extensions.definition_lists(
11210 options.tightLists)
11211 table.insert(extensions, definition_lists_extension)
11212 end
11213
11214 if options.fencedCode then
11215 local fenced_code_extension = M.extensions.fenced_code(
11216 options.blankBeforeCodeFence,
11217 options.fencedCodeAttributes,
11218 options.rawAttribute)
11219 table.insert(extensions, fenced_code_extension)
11220 end
11221
11222 if options.fencedDivs then
11223 local fenced_div_extension = M.extensions.fenced_divs(
11224 options.blankBeforeDivFence)
11225 table.insert(extensions, fenced_div_extension)
11226 end
11227
11228 if options.headerAttributes then
11229 local header_attributes_extension = M.extensions.header_attributes()
11230 table.insert(extensions, header_attributes_extension)
11231 end
11232
11233 if options.inlineCodeAttributes then
11234 local inline_code_attributes_extension =
11235 M.extensions.inline_code_attributes()
11236 table.insert(extensions, inline_code_attributes_extension)
11237 end
11238
11239 if options.jekyllData then

```

```

11240 local jekyll_data_extension = M.extensions.jekyll_data(
11241 options.expectJekyllData)
11242 table.insert(extensions, jekyll_data_extension)
11243 end
11244
11245 if options.linkAttributes then
11246 local link_attributes_extension =
11247 M.extensions.link_attributes()
11248 table.insert(extensions, link_attributes_extension)
11249 end
11250
11251 if options.lineBlocks then
11252 local line_block_extension = M.extensions.line_blocks()
11253 table.insert(extensions, line_block_extension)
11254 end
11255
11256 if options.mark then
11257 local mark_extension = M.extensions.mark()
11258 table.insert(extensions, mark_extension)
11259 end
11260
11261 if options.pipeTables then
11262 local pipe_tables_extension = M.extensions.pipe_tables(
11263 options.tableCaptions, options.tableAttributes)
11264 table.insert(extensions, pipe_tables_extension)
11265 end
11266
11267 if options.rawAttribute then
11268 local raw_inline_extension = M.extensions.raw_inline()
11269 table.insert(extensions, raw_inline_extension)
11270 end
11271
11272 if options.strikeThrough then
11273 local strike_through_extension = M.extensions.strike_through()
11274 table.insert(extensions, strike_through_extension)
11275 end
11276
11277 if options.subscripts then
11278 local subscript_extension = M.extensions.subscripts()
11279 table.insert(extensions, subscript_extension)
11280 end
11281
11282 if options.superscripts then
11283 local superscript_extension = M.extensions.superscripts()
11284 table.insert(extensions, superscript_extension)
11285 end
11286

```

```

11287 if options.texMathDollars or
11288 options.texMathSingleBackslash or
11289 options.texMathDoubleBackslash then
11290 local tex_math_extension = M.extensions.tex_math(
11291 options.texMathDollars,
11292 options.texMathSingleBackslash,
11293 options.texMathDoubleBackslash)
11294 table.insert(extensions, tex_math_extension)
11295 end
11296
11297 if options.notes or options.inlineNotes then
11298 local notes_extension = M.extensions.notes(
11299 options.notes, options.inlineNotes)
11300 table.insert(extensions, notes_extension)
11301 end
11302
11303 if options.citations then
11304 local citations_extension = M.extensions.citations(options.citationNbsps)
11305 table.insert(extensions, citations_extension)
11306 end
11307
11308 if options.fancyLists then
11309 local fancy_lists_extension = M.extensions.fancy_lists()
11310 table.insert(extensions, fancy_lists_extension)
11311 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

11312 for _, user_extension_filename in ipairs(options.extensions) do
11313 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

11314 local pathname = kpse.lookup(filename)
11315 local input_file = assert(io.open(pathname, "r"),
11316 [[Could not open user-defined syntax extension "]]
11317 .. pathname .. [[" for reading]])
11318 local input = assert(input_file:read("*a"))
11319 assert(input_file:close())
11320 local user_extension, err = load([[
11321 local sandbox = {}
11322 setmetatable(sandbox, {__index = _G})
11323 _ENV = sandbox
11324]] .. input)()
11325 assert(user_extension,
11326 [[Failed to compile user-defined syntax extension "]]
11327 .. pathname .. [[:]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

11328 assert(user_extension.api_version ~= nil,

```

```

11329 [[User-defined syntax extension "]] .. pathname
11330 .. [[" does not specify mandatory field "api_version"]])
11331 assert(type(user_extension.api_version) == "number",
11332 [[User-defined syntax extension "]] .. pathname
11333 .. [[" specifies field "api_version" of type "]]
11334 .. type(user_extension.api_version)
11335 .. [[" but "number" was expected]])
11336 assert(user_extension.api_version > 0
11337 and user_extension.api_version <= metadata.user_extension_api_version,
11338 [[User-defined syntax extension "]] .. pathname
11339 .. [[" uses syntax extension API version "]]
11340 .. user_extension.api_version .. [[but markdown.lua]]
11341 .. metadata.version .. [[uses API version]]
11342 .. metadata.user_extension_api_version
11343 .. [[, which is incompatible]])
11344
11345 assert(user_extension.grammar_version ~= nil,
11346 [[User-defined syntax extension "]] .. pathname
11347 .. [[" does not specify mandatory field "grammar_version"]])
11348 assert(type(user_extension.grammar_version) == "number",
11349 [[User-defined syntax extension "]] .. pathname
11350 .. [[" specifies field "grammar_version" of type "]]
11351 .. type(user_extension.grammar_version)
11352 .. [[" but "number" was expected]])
11353 assert(user_extension.grammar_version == metadata.grammar_version,
11354 [[User-defined syntax extension "]] .. pathname
11355 .. [[" uses grammar version "]] .. user_extension.grammar_version
11356 .. [[but markdown.lua]] .. metadata.version
11357 .. [[uses grammar version]] .. metadata.grammar_version
11358 .. [[, which is incompatible]])
11359
11360 assert(user_extension.finalize_grammar ~= nil,
11361 [[User-defined syntax extension "]] .. pathname
11362 .. [[" does not specify mandatory "finalize_grammar" field]])
11363 assert(type(user_extension.finalize_grammar) == "function",
11364 [[User-defined syntax extension "]] .. pathname
11365 .. [[" specifies field "finalize_grammar" of type "]]
11366 .. type(user_extension.finalize_grammar)
11367 .. [[" but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```

11368 local extension = {
11369 name = [[user-defined "]] .. pathname .. [[" syntax extension]],
11370 extend_reader = user_extension.finalize_grammar,
11371 extend_writer = function() end,
11372 }

```



```

11373 return extension
11374 end)(user_extension_filename)
11375 table.insert(extensions, user_extension)
11376 end

```

Produce a conversion function from markdown to plain  $\text{\TeX}$ .

```

11377 local writer = M.writer.new(options)
11378 local reader = M.reader.new(writer, options)
11379 local convert = reader.finalize_grammar(extensions)

```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```

11380 collectgarbage("collect")

```

Update the singleton cache.

```

11381 if options.singletonCache then
11382 local singletonCacheOptions = {}
11383 for k, v in pairs(options) do
11384 singletonCacheOptions[k] = v
11385 end
11386 setmetatable(singletonCacheOptions,
11387 { __index = function (_, key)
11388 return defaultOptions[key] end })
11389 singletonCache.options = singletonCacheOptions
11390 singletonCache.convert = convert
11391 end

```

Return the conversion function from markdown to plain  $\text{\TeX}$ .

```

11392 return convert
11393 end
11394
11395 return M

```

### 3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

11396
11397 local input
11398 if input_filename then
11399 local input_file = assert(io.open(input_filename, "r"),
11400 [[Could not open file]] .. input_filename .. [[for reading]])
11401 input = assert(input_file:read("*a"))
11402 assert(input_file:close())
11403 else
11404 input = assert(io.read("*a"))
11405 end
11406

```

First, ensure that the `options.cacheDir` directory exists.

```
11407 local lfs = require("lfs")
11408 if options.cacheDir and not lfs.isdir(options.cacheDir) then
11409 assert(lfs.mkdir(options["cacheDir"]))
11410 end
```

If Kpathsea has not been loaded before or if Lua<sub>TEX</sub> has not yet been initialized, configure Kpathsea on top of loading it.

```
11411 local kpse
11412 (function()
11413 local should_initialize = package.loaded.kpse == nil
11414 or tex.initialize ~= nil
11415 kpse = require("kpse")
11416 if should_initialize then
11417 kpse.set_program_name("luatex")
11418 end
11419 end)()
11420 local md = require("markdown")
```

Since we are loading the rest of the Lua implementation dynamically, check that both the `markdown` module and the command line implementation are the same version.

```
11421 if metadata.version ~= md.metadata.version then
11422 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
11423 "markdown.lua " .. md.metadata.version .. ".")
11424 end
11425 local convert = md.new(options)
11426 local output = convert(input)
11427
11428 if output_filename then
11429 local output_file = assert(io.open(output_filename, "w"),
11430 [[Could not open file]] .. output_filename .. [[for writing]])
11431 assert(output_file:write(output))
11432 assert(output_file:close())
11433 else
11434 assert(io.write(output))
11435 end
```

Remove the `options.cacheDir` directory if it is empty.

```
11436 if options.cacheDir then
11437 lfs.rmdir(options["cacheDir"])
11438 end
```

### 3.2 Plain $\text{T}_{\text{E}}\text{X}$ Implementation

The plain  $\text{T}_{\text{E}}\text{X}$  implementation provides macros for the interfacing between  $\text{T}_{\text{E}}\text{X}$  and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain  $\text{T}_{\text{E}}\text{X}$  exposed by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Section 2.2).

### 3.2.1 Logging Facilities

```
11439 \ExplSyntaxOn
11440 \cs_if_free:NT
11441 \markdownInfo
11442 {
11443 \cs_new:Npn
11444 \markdownInfo #1
11445 {
11446 \msg_info:nne
11447 { markdown }
11448 { generic-message }
11449 { #1 }
11450 }
11451 }
11452 \cs_if_free:NT
11453 \markdownWarning
11454 {
11455 \cs_new:Npn
11456 \markdownWarning #1
11457 {
11458 \msg_warning:nne
11459 { markdown }
11460 { generic-message }
11461 { #1 }
11462 }
11463 }
11464 \cs_if_free:NT
11465 \markdownError
11466 {
11467 \cs_new:Npn
11468 \markdownError #1 #2
11469 {
11470 \msg_error:nnee
11471 { markdown }
11472 { generic-message-with-help-text }
11473 { #1 }
11474 { #2 }
11475 }
11476 }
11477 \msg_new:nnn
11478 { markdown }
11479 { generic-message }
11480 { #1 }
11481 \msg_new:nnnn
11482 { markdown }
11483 { generic-message-with-help-text }
```

```

11484 { #1 }
11485 { #2 }
11486 \cs_generate_variant:Nn
11487 \msg_info:nnn
11488 { nne }
11489 \cs_generate_variant:Nn
11490 \msg_warning:nnn
11491 { nne }
11492 \cs_generate_variant:Nn
11493 \msg_error:nnnn
11494 { nnee }
11495 \ExplSyntaxOff

```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain  $\TeX$  themes provided with the Markdown package.

```

11496 \ExplSyntaxOn
11497 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
11498 \cs_new:Nn
11499 \@@_plain_tex_load_theme:nn
11500 {
11501 \prop_get:NnNTF
11502 \g_@@_plain_tex_loaded_themes_linenos_prop
11503 { #1 }
11504 \l_tmpa_tl
11505 {
11506 \msg_warning:nnnV
11507 { markdown }
11508 { repeatedly-loaded-plain-tex-theme }
11509 { #1 }
11510 \l_tmpa_tl
11511 }
11512 {
11513 \msg_info:nnn
11514 { markdown }
11515 { loading-plain-tex-theme }
11516 { #1 }
11517 \prop_gput:Nnx
11518 \g_@@_plain_tex_loaded_themes_linenos_prop
11519 { #1 }
11520 { \tex_the:D \tex_inputlineno:D }
11521 \file_input:n
11522 { markdown theme #2 }
11523 }
11524 }

```

```

11525 \msg_new:nnn
11526 { markdown }
11527 { loading-plain-tex-theme }
11528 { Loading~plain~TeX~Markdown~theme~#1 }
11529 \msg_new:nnn
11530 { markdown }
11531 { repeatedly-loaded-plain-tex-theme }
11532 {
11533 Plain~TeX~Markdown~theme~#1~was~previously~
11534 loaded~on~line~#2,~not~loading~it~again
11535 }
11536 \cs_generate_variant:Nn
11537 \prop_gput:Nnn
11538 { Nnx }
11539 \cs_gset_eq:NN
11540 \@@_load_theme:nn
11541 \@@_plain_tex_load_theme:nn
11542 \cs_generate_variant:Nn
11543 \@@_load_theme:nn
11544 { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain  $\TeX$  theme from within themes for higher-level  $\TeX$  formats such as  $\LaTeX$  and  $\ConTeXt$ .

```

11545 \cs_new:Npn
11546 \markdownLoadPlainTeXTheme
11547 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

11548 \tl_set:NV
11549 \l_tmpa_tl
11550 \g_@@_current_theme_tl
11551 \tl_reverse:N
11552 \l_tmpa_tl
11553 \tl_set:Ne
11554 \l_tmpb_tl
11555 {
11556 \tl_tail:V
11557 \l_tmpa_tl
11558 }
11559 \tl_reverse:N
11560 \l_tmpb_tl

```

Next, we munge the theme name.

```

11561 \str_set:NV
11562 \l_tmpa_str
11563 \l_tmpb_tl

```

```

11564 \str_replace_all:Nnn
11565 \l_tmpa_str
11566 { / }
11567 { _ }

```

Finally, we load the plain TeX theme.

```

11568 \@_plain_tex_load_theme:VV
11569 \l_tmpb_tl
11570 \l_tmpa_str
11571 }
11572 \cs_generate_variant:Nn
11573 \tl_set:Nn
11574 { Ne }
11575 \cs_generate_variant:Nn
11576 \@_plain_tex_load_theme:nn
11577 { VV }
11578 \ExplSyntaxOff

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

11579 \markdownSetup {
11580 rendererPrototypes = {
11581 tilde = {~},
11582 },
11583 }

```

The [witiko/markdown/defaults](#) plain TeX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

11584 \def\markdownRendererInterblockSeparatorPrototype{\par}%
11585 \def\markdownRendererParagraphSeparatorPrototype{%
11586 \markdownRendererInterblockSeparator}%
11587 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
11588 \def\markdownRendererSoftLineBreakPrototype{ }%
11589 \let\markdownRendererEllipsisPrototype\dots
11590 \def\markdownRendererNbspPrototype{~}%
11591 \def\markdownRendererLeftBracePrototype{\char`\\}%
11592 \def\markdownRendererRightBracePrototype{\char`\}}%
11593 \def\markdownRendererDollarSignPrototype{\char`$}%
11594 \def\markdownRendererPercentSignPrototype{\char`\}%
11595 \def\markdownRendererAmpersandPrototype{\&%
11596 \def\markdownRendererUnderscorePrototype{\char`_}%
11597 \def\markdownRendererHashPrototype{\char`#}%
11598 \def\markdownRendererCircumflexPrototype{\char`^}%
11599 \def\markdownRendererBackslashPrototype{\char`\}%

```

```

11600 \def\markdownRendererTildePrototype{\char`~}%
11601 \def\markdownRendererPipePrototype{|}%
11602 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
11603 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
11604 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
11605 \markdownInput{#3}}%
11606 \def\markdownRendererContentBlockOnlineImagePrototype{%
11607 \markdownRendererImage}%
11608 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
11609 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
11610 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
11611 \def\markdownRendererUlBeginPrototype{}%
11612 \def\markdownRendererUlBeginTightPrototype{}%
11613 \def\markdownRendererUlItemPrototype{}%
11614 \def\markdownRendererUlItemEndPrototype{}%
11615 \def\markdownRendererUlEndPrototype{}%
11616 \def\markdownRendererUlEndTightPrototype{}%
11617 \def\markdownRendererOlBeginPrototype{}%
11618 \def\markdownRendererOlBeginTightPrototype{}%
11619 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
11620 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
11621 \def\markdownRendererOlItemPrototype{}%
11622 \def\markdownRendererOlItemWithNumberPrototype#1{}%
11623 \def\markdownRendererOlItemEndPrototype{}%
11624 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
11625 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
11626 \def\markdownRendererFancyOlItemEndPrototype{}%
11627 \def\markdownRendererOlEndPrototype{}%
11628 \def\markdownRendererOlEndTightPrototype{}%
11629 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
11630 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
11631 \def\markdownRendererDlBeginPrototype{}%
11632 \def\markdownRendererDlBeginTightPrototype{}%
11633 \def\markdownRendererDlItemPrototype#1{#1}%
11634 \def\markdownRendererDlItemEndPrototype{}%
11635 \def\markdownRendererDlDefinitionBeginPrototype{}%
11636 \def\markdownRendererDlDefinitionEndPrototype{\par}%
11637 \def\markdownRendererDlEndPrototype{}%
11638 \def\markdownRendererDlEndTightPrototype{}%
11639 \def\markdownRendererEmphasisPrototype#1{\it#1}%
11640 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
11641 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
11642 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
11643 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
11644 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
11645 \def\markdownRendererInputVerbatimPrototype#1{%
11646 \par{\tt\input#1\relax{}}\par}%

```

```

11647 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
11648 \markdownRendererInputVerbatim{#1}}%
11649 \def\markdownRendererHeadingOnePrototype#1{#1}%
11650 \def\markdownRendererHeadingTwoPrototype#1{#1}%
11651 \def\markdownRendererHeadingThreePrototype#1{#1}%
11652 \def\markdownRendererHeadingFourPrototype#1{#1}%
11653 \def\markdownRendererHeadingFivePrototype#1{#1}%
11654 \def\markdownRendererHeadingSixPrototype#1{#1}%
11655 \def\markdownRendererThematicBreakPrototype{}%
11656 \def\markdownRendererNotePrototype#1{#1}%
11657 \def\markdownRendererCitePrototype#1{}%
11658 \def\markdownRendererTextCitePrototype#1{}%
11659 \def\markdownRendererTickedBoxPrototype{[X]}%
11660 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
11661 \def\markdownRendererUntickedBoxPrototype{[]}%
11662 \def\markdownRendererStrikeThroughPrototype#1{#1}%
11663 \def\markdownRendererSuperscriptPrototype#1{#1}%
11664 \def\markdownRendererSubscriptPrototype#1{#1}%
11665 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
11666 \def\markdownRendererInlineMathPrototype#1{$#1$}%
11667 \ExplSyntaxOn
11668 \cs_gset:Npn
11669 \markdownRendererHeaderAttributeContextBeginPrototype
11670 {
11671 \group_begin:
11672 \color_group_begin:
11673 }
11674 \cs_gset:Npn
11675 \markdownRendererHeaderAttributeContextEndPrototype
11676 {
11677 \color_group_end:
11678 \group_end:
11679 }
11680 \cs_gset_eq:NN
11681 \markdownRendererBracketedSpanAttributeContextBeginPrototype
11682 \markdownRendererHeaderAttributeContextBeginPrototype
11683 \cs_gset_eq:NN
11684 \markdownRendererBracketedSpanAttributeContextEndPrototype
11685 \markdownRendererHeaderAttributeContextEndPrototype
11686 \cs_gset_eq:NN
11687 \markdownRendererFencedDivAttributeContextBeginPrototype
11688 \markdownRendererHeaderAttributeContextBeginPrototype
11689 \cs_gset_eq:NN
11690 \markdownRendererFencedDivAttributeContextEndPrototype
11691 \markdownRendererHeaderAttributeContextEndPrototype
11692 \cs_gset_eq:NN
11693 \markdownRendererFencedCodeAttributeContextBeginPrototype

```



```

11694 \markdownRendererHeaderAttributeContextBeginPrototype
11695 \cs_gset_eq:NN
11696 \markdownRendererFencedCodeAttributeContextEndPrototype
11697 \markdownRendererHeaderAttributeContextEndPrototype
11698 \cs_gset:Npn
11699 \markdownRendererReplacementCharacterPrototype
11700 { \codepoint_str_generate:n { fffd } }
11701 \ExplSyntaxOff
11702 \def\markdownRendererSectionBeginPrototype{}%
11703 \def\markdownRendererSectionEndPrototype{}%

```

### 3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

11704 \ExplSyntaxOn
11705 \cs_new:Nn
11706 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11707 {
11708 \str_case:nn
11709 { #2 }
11710 {
11711 { md } { \markdownInput{#1} }
11712 { tex } { \markdownEscape{#1} \unskip }
11713 }
11714 }
11715 \cs_new:Nn
11716 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11717 {
11718 \str_case:nn
11719 { #2 }
11720 {
11721 { md } { \markdownInput{#1} }
11722 { tex } { \markdownEscape{#1} }
11723 }
11724 }
11725 \cs_gset:Npn
11726 \markdownRendererInputRawInlinePrototype#1#2
11727 {
11728 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11729 { #1 }
11730 { #2 }
11731 }
11732 \cs_gset:Npn
11733 \markdownRendererInputRawBlockPrototype#1#2
11734 {

```

```

11735 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11736 { #1 }
11737 { #2 }
11738 }
11739 \ExplSyntaxOff

```

### 3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

11740 \ExplSyntaxOn
11741 \seq_new:N \g_@@_jekyll_data_datatypes_seq
11742 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
11743 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
11744 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```

11745 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
11746 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
11747 {
11748 \seq_if_empty:NF
11749 \g_@@_jekyll_data_datatypes_seq
11750 {
11751 \seq_get_right:NN
11752 \g_@@_jekyll_data_datatypes_seq
11753 \l_tmpa_tl

```

If we are currently in a sequence, we will put an asterisk (\*) instead of a key into `\g_@@_jekyll_data_wildcard_absolute_address_seq` to make it represent a *wildcard*. Keeping a wildcard instead of a precise address makes it easy for the users to react to *any* item of a sequence regardless of how many there are, which can often be useful.

```

11754 \str_if_eq:NNTF

```

```

11755 \l_tmpa_tl
11756 \c_@@_jekyll_data_sequence_tl
11757 {
11758 \seq_put_right:Nn
11759 \g_@@_jekyll_data_wildcard_absolute_address_seq
11760 { * }
11761 }
11762 {
11763 \seq_put_right:Nn
11764 \g_@@_jekyll_data_wildcard_absolute_address_seq
11765 { #1 }
11766 }
11767 }
11768 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```

11769 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
11770 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
11771 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
11772 {
11773 \seq_pop_left:NN #1 \l_tmpa_tl

```

```

11774 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
11775 \seq_put_left:NV #1 \l_tmpa_tl
11776 }
11777 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
11778 {
11779 \markdown_jekyll_data_concatenate_address:NN
11780 \g_@@_jekyll_data_wildcard_absolute_address_seq
11781 \g_@@_jekyll_data_wildcard_absolute_address_tl
11782 \seq_get_right:NN
11783 \g_@@_jekyll_data_wildcard_absolute_address_seq
11784 \g_@@_jekyll_data_wildcard_relative_address_tl
11785 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```

11786 \cs_new:Nn \markdown_jekyll_data_push:nN
11787 {
11788 \markdown_jekyll_data_push_address_segment:n
11789 { #1 }
11790 \seq_put_right:NV
11791 \g_@@_jekyll_data_datatypes_seq
11792 #2
11793 \markdown_jekyll_data_update_address_tls:
11794 }
11795 \cs_new:Nn \markdown_jekyll_data_pop:
11796 {
11797 \seq_pop_right:NN
11798 \g_@@_jekyll_data_wildcard_absolute_address_seq
11799 \l_tmpa_tl
11800 \seq_pop_right:NN
11801 \g_@@_jekyll_data_datatypes_seq
11802 \l_tmpa_tl
11803 \markdown_jekyll_data_update_address_tls:
11804 }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

11805 \cs_new:Nn \markdown_jekyll_data_set_keyval:nN
11806 {
11807 \keys_set_known:nn
11808 { markdown/jekyllData }
11809 { { #1 } = { #2 } }
11810 }
11811 \cs_generate_variant:Nn
11812 \markdown_jekyll_data_set_keyval:nN
11813 { Vn }
11814 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn

```

```

11815 {
11816 \markdown_jekyll_data_push:nN
11817 { #1 }
11818 \c_@@_jekyll_data_scalar_tl
11819 \markdown_jekyll_data_set_keyval:Vn
11820 \g_@@_jekyll_data_wildcard_absolute_address_tl
11821 { #2 }
11822 \markdown_jekyll_data_set_keyval:Vn
11823 \g_@@_jekyll_data_wildcard_relative_address_tl
11824 { #2 }
11825 \markdown_jekyll_data_pop:
11826 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

11827 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
11828 \markdown_jekyll_data_push:nN
11829 { #1 }
11830 \c_@@_jekyll_data_sequence_tl
11831 }
11832 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
11833 \markdown_jekyll_data_push:nN
11834 { #1 }
11835 \c_@@_jekyll_data_mapping_tl
11836 }
11837 \def\markdownRendererJekyllDataSequenceEndPrototype{
11838 \markdown_jekyll_data_pop:
11839 }
11840 \def\markdownRendererJekyllDataMappingEndPrototype{
11841 \markdown_jekyll_data_pop:
11842 }
11843 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
11844 \markdown_jekyll_data_set_keyvals:nn
11845 { #1 }
11846 { #2 }
11847 }
11848 \def\markdownRendererJekyllDataEmptyPrototype#1{}
11849 \def\markdownRendererJekyllDataNumberPrototype#1#2{
11850 \markdown_jekyll_data_set_keyvals:nn
11851 { #1 }
11852 { #2 }
11853 }
11854 \def\markdownRendererJekyllDataStringPrototype#1#2{
11855 \markdown_jekyll_data_set_keyvals:nn
11856 { #1 }
11857 { #2 }
11858 }

```

```
11859 \ExplSyntaxOff
```

If plain T<sub>E</sub>X is the top layer, we load the `witiko/markdown/defaults` plain T<sub>E</sub>X theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
11860 \ExplSyntaxOn
11861 \str_if_eq:VVT
11862 \c_@@_top_layer_tl
11863 \c_@@_option_layer_plain_tex_tl
11864 {
11865 \ExplSyntaxOff
11866 \@@_if_option:nF
11867 { noDefaults }
11868 {
11869 \@@_setup:n
11870 {theme = witiko/markdown/defaults}
11871 }
11872 \ExplSyntaxOn
11873 }
11874 \ExplSyntaxOff
```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain T<sub>E</sub>X options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```
11875 \ExplSyntaxOn
11876 \tl_new:N \g_@@_formatted_lua_options_tl
11877 \cs_new:Nn \@@_format_lua_options:
11878 {
11879 \tl_gclear:N
11880 \g_@@_formatted_lua_options_tl
11881 \seq_map_function:NN
11882 \g_@@_lua_options_seq
11883 \@@_format_lua_option:n
11884 }
11885 \cs_new:Nn \@@_format_lua_option:n
11886 {
11887 \@@_typecheck_option:n
11888 { #1 }
11889 \@@_get_option_type:nN
11890 { #1 }
11891 \l_tmpa_tl
11892 \bool_case_true:nF
11893 {
11894 {
11895 \str_if_eq_p:VV
```

```

11896 \l_tmpa_tl
11897 \c_@@_option_type_boolean_tl ||
11898 \str_if_eq_p:VV
11899 \l_tmpa_tl
11900 \c_@@_option_type_number_tl ||
11901 \str_if_eq_p:VV
11902 \l_tmpa_tl
11903 \c_@@_option_type_counter_tl
11904 }
11905 {
11906 \@@_get_option_value:nN
11907 { #1 }
11908 \l_tmpa_tl
11909 \tl_gput_right:Nx
11910 \g_@@_formatted_lua_options_tl
11911 { #1~::~ \l_tmpa_tl ,~ }
11912 }
11913 {
11914 \str_if_eq_p:VV
11915 \l_tmpa_tl
11916 \c_@@_option_type_clist_tl
11917 }
11918 {
11919 \@@_get_option_value:nN
11920 { #1 }
11921 \l_tmpa_tl
11922 \tl_gput_right:Nx
11923 \g_@@_formatted_lua_options_tl
11924 { #1~::~\c_left_brace_str }
11925 \clist_map_inline:Vn
11926 \l_tmpa_tl
11927 {
11928 \tl_gput_right:Nx
11929 \g_@@_formatted_lua_options_tl
11930 { "##1" ,~ }
11931 }
11932 \tl_gput_right:Nx
11933 \g_@@_formatted_lua_options_tl
11934 { \c_right_brace_str ,~ }
11935 }
11936 }
11937 {
11938 \@@_get_option_value:nN
11939 { #1 }
11940 \l_tmpa_tl
11941 \tl_gput_right:Nx
11942 \g_@@_formatted_lua_options_tl

```

```

11943 { #1~::~ " \l_tmpa_tl " ,~ }
11944 }
11945 }
11946 \cs_generate_variant:Nn
11947 \clist_map_inline:nn
11948 { Vn }
11949 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
11950 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
11951 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain TeX. It exposes the `convert` function for the use by any further Lua code.

```

11952 \def\markdownPrepare{%
First, ensure that the cacheDir directory exists.
11953 local lfs = require("lfs")
11954 local cacheDir = "\markdownOptionCacheDir"
11955 if not lfs.isdir(cacheDir) then
11956 assert(lfs.mkdir(cacheDir))
11957 end

```

Next, load the `markdown` module and create a converter function using the plain TeX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

11958 local md = require("markdown")
11959 local convert = md.new(\markdownLuaOptions)
11960 }%

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TeX.

```

11961 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
11962 lfs.rmdir(cacheDir)
11963 }%

```

### 3.2.5 Buffering Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```

11964 \csname newread\endcsname\markdownInputFileStream
11965 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

11966 \begingroup
11967 \catcode\^^I=12%
11968 \gdef\markdownReadAndConvertTab{^^I}%
11969 \endgroup

```



The `\markdownReadAndConvert` macro is largely a rewrite of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> `\filecontents` macro to plain T<sub>E</sub>X.

```
11970 \begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `stripPercentSigns` is enabled.

```
11971 \catcode\^^M=13%
11972 \catcode\^^I=13%
11973 \catcode|=0%
11974 \catcode\\=12%
11975 |catcode@=14%
11976 |catcode%=12@
11977 |gdef|markdownReadAndConvert#1#2{@
11978 |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
11979 |markdownIfOption{frozenCache}{-}{@
11980 |immediate|openout|markdownOutputFileStream@
11981 |markdownOptionInputTempFileName|relax@
11982 |markdownInfo{Buffering markdown input into the temporary @
11983 input file "|markdownOptionInputTempFileName" and scanning @
11984 for the closing token sequence "#1"}@
11985 }@
```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
11986 |def|do##1{|catcode`##1=12}|dospecials@
11987 |catcode`=12@
11988 |markdownMakeOther@
```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (`%`) when `stripPercentSigns` is enabled. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```
11989 |def|markdownReadAndConvertStripPercentSign##1{@
11990 |markdownIfOption{stripPercentSigns}{@
11991 |if##1%@
11992 |expandafter|expandafter|expandafter@
11993 |markdownReadAndConvertProcessLine@
11994 |else@
11995 |expandafter|expandafter|expandafter@
11996 |markdownReadAndConvertProcessLine@
```

```

11997 |expandafter|expandafter|expandafter##1@
11998 |fi@
11999 }{@
12000 |expandafter@
12001 |markdownReadAndConvertProcessLine@
12002 |expandafter##1@
12003 }@
12004 }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

12005 |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

12006 |ifx|relax##3|relax@
12007 |markdownIfOption{frozenCache}{}{@
12008 |immediate|write|markdownOutputFileStream{##1}@
12009 }@
12010 |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain TEX, `\input` the result of the conversion, and expand the ending control sequence.

```

12011 |def^^M{@
12012 |markdownInfo{The ending token sequence was found}@
12013 |markdownIfOption{frozenCache}{}{@
12014 |immediate|closeout|markdownOutputFileStream@
12015 }@
12016 |endgroup@
12017 |markdownInput{@
12018 |markdownOptionOutputDir@
12019 /|markdownOptionInputTempFileName@
12020 }@
12021 #2}@
12022 |fi@

```

Repeat with the next line.

```

12023 ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

12024 |catcode`|^I=13@
12025 |def^^I{|markdownReadAndConvertTab}@

```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```

12026 |catcode`|^M=13@
12027 |def^M##1^M{@
12028 |def^M###1^M{@
12029 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
12030 ^M}@
12031 ^M}@

```

Reset the character categories back to the former state.

```
12032 |endgroup
```

Use the `lt3luabridge` library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```

12033 \ExplSyntaxOn
12034 \cs_new:Npn
12035 \markdownLuaExecute
12036 #1
12037 {
12038 \int_compare:nNt
12039 { \g_luabridge_method_int }
12040 =
12041 { \c_luabridge_method_shell_int }
12042 {
12043 \sys_if_shell_unrestricted:F
12044 {
12045 \sys_if_shell:TF
12046 {
12047 \msg_error:nn
12048 { markdown }
12049 { restricted-shell-access }
12050 }
12051 {
12052 \msg_error:nn
12053 { markdown }
12054 { disabled-shell-access }
12055 }
12056 }
12057 }
12058 \luabridge_now:e
12059 { #1 }
12060 }
12061 \cs_generate_variant:Nn
12062 \msg_new:nnnn
12063 { nnnV }
12064 \tl_set:Nn

```

```

12065 \l_tmpa_tl
12066 {
12067 You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
12068 --enable-write18~flag,~or~write~shell_escape=t~in~the~
12069 texmf.cnf~file.
12070 }
12071 \msg_new:nnnV
12072 { markdown }
12073 { restricted-shell-access }
12074 { Shell-escape-is-restricted }
12075 \l_tmpa_tl
12076 \msg_new:nnnV
12077 { markdown }
12078 { disabled-shell-access }
12079 { Shell-escape-is-disabled }
12080 \l_tmpa_tl
12081 \ExplSyntaxOff

```

### 3.2.6 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TeX.

```
12082 \begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code. Furthermore, use the ampersand symbol to specify parameters.

```

12083 \catcode`\|=0%
12084 \catcode`\|=12%
12085 \catcode`\&=6%
12086 |gdef|markdownInput#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

12087 |begingroup
12088 |catcode`\|=12

```

Furthermore, also change the category code of the hash sign (#) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```
12089 |catcode`\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
12090 |markdownIfOption{frozenCache}{%
```

```

12091 |ifnum|markdownOptionFrozenCacheCounter=0|relax
12092 |markdownInfo{Reading frozen cache from
12093 |"|markdownOptionFrozenCacheFileName"}%
12094 |input|markdownOptionFrozenCacheFileName|relax
12095 |fi
12096 |markdownInfo{Including markdown document number
12097 |"|the|markdownOptionFrozenCacheCounter" from frozen cache}%
12098 |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
12099 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
12100 }-%
12101 |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

12102 |openin|markdownInputFileStream&1
12103 |closein|markdownInputFileStream
12104 |markdownPrepareLuaOptions
12105 |markdownLuaExecute{%
12106 |markdownPrepare
12107 local file = assert(io.open("&1", "r"),
12108 [[Could not open file "&1" for reading]])
12109 local input = assert(file:read("*a"))
12110 assert(file:close())
12111 print(convert(input))
12112 |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

12113 |markdownIfOption{finalizeCache}{-%
12114 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{-%
12115 }-%
12116 |endgroup
12117 }-%
12118 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of T<sub>E</sub>X to execute a T<sub>E</sub>X document in the middle of a markdown document fragment.

```

12119 \gdef\markdownEscape#1{%
12120 \catcode`\%=14\relax
12121 \catcode`\#=6\relax
12122 \input #1\relax
12123 \catcode`\%=12\relax
12124 \catcode`\#=12\relax
12125 }%

```

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implementation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [11, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```
12126 \def\markdownVersionSpace{ }%
12127 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
12128 \markdownVersion\markdownVersionSpace markdown renderer]%
```

#### 3.3.1 Logging Facilities

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

#### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.2).

```
12129 \let\markdownInputPlainTeX\markdownInput
12130 \renewcommand\markdownInput [2] []{%
12131 \begingroup
12132 \markdownSetup{#1}%
12133 \markdownInputPlainTeX{#2}%
12134 \endgroup}%
```

The `markdown`, and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```
12135 \ExplSyntaxOn
12136 \renewenvironment
12137 { markdown }
12138 {
```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

|                                                                                                   |                                                                             |
|---------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <pre>\begin{markdown} [smartEllipses] % This is an optional argument ^ % ... \end{markdown}</pre> | <pre>\begin{markdown} [smartEllipses] % ^ This is link \end{markdown}</pre> |
|---------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
12139 \group_begin:
12140 \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
12141 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
12142 \peek_regex_replace_once:nnF
12143 { \ *[\r*([~]]*)\][^~\r]* }
12144 {
```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce `\par` tokens. Furthermore, this will cause hash signs followed by a number to be recognized as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.

```
12145 \c { group_end: }
12146 \c { t1_set_rescan:Nnn } \c { l_tmpa_t1 } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
12147 \c { @@_setup:V } \c { l_tmpa_t1 }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the L<sup>A</sup>T<sub>E</sub>X environment.

```
12148 \c { markdownReadAndConvert@markdown } { }
12149 }
12150 {
12151 \group_end:
12152 \markdownReadAndConvert@markdown { }
12153 }
12154 }
12155 { \markdownEnd }
12156 \renewenvironment
12157 { markdown* }
```

```

12158 [1]
12159 {
12160 \msg_warning:nnn
12161 { markdown }
12162 { latex-markdown-star-deprecated }
12163 { #1 }
12164 \@@_setup:n
12165 { #1 }
12166 \markdownReadAndConvert@markdown *
12167 }
12168 { \markdownEnd }
12169 \msg_new:nnn
12170 { markdown }
12171 { latex-markdown-star-deprecated }
12172 {
12173 The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
12174 be~removed~in~the~next~major~version~of~the~Markdown~package.
12175 }
12176 \ExplSyntaxOff
12177 \begingroup

```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```

12178 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
12179 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
12180 |gdef|markdownReadAndConvert@markdown#1<%
12181 |markdownReadAndConvert<\end{markdown#1}>%
12182 <|end<markdown#1>>>%
12183 |endgroup

```

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

12184 \DeclareOption*{%
12185 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
12186 \ProcessOptions\relax

```

### 3.3.4 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

12187 \ExplSyntaxOn
12188 \cs_gset:Nn

```



```

12189 \@@_load_theme:nn
12190 {

```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

12191 \ifmarkdownLaTeXLoaded
12192 \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load L<sup>A</sup>T<sub>E</sub>X themes after the preamble. Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

12193 \file_if_exist:nTF
12194 { markdown theme #2.sty }
12195 {
12196 \msg_error:nnn
12197 { markdown }
12198 { latex-theme-after-preamble }
12199 { #1 }
12200 }
12201 {
12202 \@@_plain_tex_load_theme:nn
12203 { #1 }
12204 { #2 }
12205 }
12206 \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```

12207 \file_if_exist:nTF
12208 { markdown theme #2.sty }
12209 {
12210 \msg_info:nnn
12211 { markdown }
12212 { loading-latex-theme }
12213 { #1 }
12214 \RequirePackage
12215 { markdown theme #2 }
12216 }
12217 {
12218 \@@_plain_tex_load_theme:nn
12219 { #1 }
12220 { #2 }
12221 }
12222 \fi
12223 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

12224 \msg_info:nnn
12225 { markdown }
12226 { theme-loading-postponed }
12227 { #1 }
12228 \AtEndOfPackage
12229 {
12230 \@_load_theme:nn
12231 { #1 }
12232 { #2 }
12233 }
12234 \fi
12235 }
12236 \msg_new:nnn
12237 { markdown }
12238 { theme-loading-postponed }
12239 {
12240 Postponing~loading~Markdown~theme~#1~until~
12241 Markdown~package~has~finished~loading
12242 }
12243 \msg_new:nnn
12244 { markdown }
12245 { loading-latex-theme }
12246 { Loading~LaTeX~Markdown~theme~#1 }
12247 \cs_generate_variant:Nn
12248 \msg_new:nnnn
12249 { nnVV }
12250 \tl_set:Nn
12251 \l_tmpa_tl
12252 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
12253 \tl_put_right:NV
12254 \l_tmpa_tl
12255 \c_backslash_str
12256 \tl_put_right:Nn
12257 \l_tmpa_tl
12258 { begin{document} }
12259 \tl_set:Nn
12260 \l_tmpb_tl
12261 { Load~Markdown~theme~#1~before~ }
12262 \tl_put_right:NV
12263 \l_tmpb_tl
12264 \c_backslash_str
12265 \tl_put_right:Nn
12266 \l_tmpb_tl
12267 { begin{document} }
12268 \msg_new:nnVV
12269 { markdown }
12270 { latex-theme-after-preamble }

```

```

12271 \l_tmpa_tl
12272 \l_tmpb_tl
12273 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
12274 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
12275 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```

12276 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
12277 \markdownRendererInputFencedCodePrototype

```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain TeX option is disabled and the code block has not been previously typeset:

```

12278 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
12279 \def\next##1 ##2\relax{%
12280 \ifthenelse{\equal{##1}{dot}}{%
12281 \markdownIfOption{frozenCache}{}{%
12282 \immediate\write18{%
12283 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
12284 then
12285 dot -Tpdf -o #1.pdf #1;
12286 cp #1 #1.pdf.source;
12287 fi}}%

```

We include the typeset image using the image token renderer:

```
12288 \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%
```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

12289 }{%
12290 \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}{#3}%
12291 }%
12292 }%
12293 \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

12294 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
12295 \markdownRendererImagePrototype

```

We load the `catchfile` and `grffile` packages, see also Section 1.1.3:

```
12296 \RequirePackage{catchfile,grffile}
```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```
12297 \newcount\markdown@witiko@graphicx@http@counter
12298 \markdown@witiko@graphicx@http@counter=0
12299 \newcommand\markdown@witiko@graphicx@http@filename{%
12300 \markdownOptionCacheDir/witiko_graphicx_http%
12301 .\the\markdown@witiko@graphicx@http@counter}%
```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```
12302 \newcommand\markdown@witiko@graphicx@http@download[2]{%
12303 wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}
```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```
12304 \begingroup
12305 \catcode`\%=12
12306 \catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
12307 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
12308 \begingroup
12309 \edef\filename{\markdown@witiko@graphicx@http@filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```
12310 \markdownIfOption{frozenCache}{}{^^A
12311 \immediate\write18{^^A
12312 mkdir -p "\markdownOptionCacheDir";
12313 if printf '%s' "#3" | grep -q -E '^https?:';
12314 then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
12315 OUTPUT_PREFIX="\markdownOptionCacheDir";
12316 OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
12317 OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
12318 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
12319 if ! [-e "$OUTPUT"];
12320 then
12321 \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
```

```

12322 printf '%s' "$OUTPUT" > "\filename";
12323 fi;

```

If the image does not have the http or https protocols or the image has already been downloaded, the URL will be stored as-is:

```

12324 else
12325 printf '%s' '#3' > "\filename";
12326 fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

12327 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
12328 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
12329 {#1}{#2}{\filename}{#4}^^A
12330 \endgroup
12331 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
12332 \endgroup

```

The `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>X theme provides default definitions for token renderer prototypes. First, the L<sup>A</sup>T<sub>E</sub>X theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```

12333 \markdownLoadPlainTeXTheme

```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.5 for the actual definitions.

### 3.3.5 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

12334 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the paralist package.

```

12335 \@ifclassloaded{beamer}{}{
12336 \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
12337 \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
12338 }

```

If we loaded the paralist package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

12339 \ExplSyntaxOn
12340 \@ifpackageloaded{paralist}{
12341 \tl_new:N
12342 \l_@@_latex_fancy_list_item_label_number_style_tl
12343 \tl_new:N
12344 \l_@@_latex_fancy_list_item_label_delimiter_style_tl

```

```

12345 \cs_new:Nn
12346 \@@_latex_fancy_list_item_label_number:nn
12347 {
12348 \str_case:nn
12349 { #1 }
12350 {
12351 { Decimal } { #2 }
12352 { LowerRoman } { \int_to_roman:n { #2 } }
12353 { UpperRoman } { \int_to_Roman:n { #2 } }
12354 { LowerAlpha } { \int_to_alph:n { #2 } }
12355 { UpperAlpha } { \int_to_Alph:n { #2 } }
12356 }
12357 }
12358 \cs_new:Nn
12359 \@@_latex_fancy_list_item_label_delimiter:n
12360 {
12361 \str_case:nn
12362 { #1 }
12363 {
12364 { Default } { . }
12365 { OneParen } {) }
12366 { Period } { . }
12367 }
12368 }
12369 \cs_new:Nn
12370 \@@_latex_fancy_list_item_label:nnn
12371 {
12372 \@@_latex_fancy_list_item_label_number:nn
12373 { #1 }
12374 { #3 }
12375 \@@_latex_fancy_list_item_label_delimiter:n
12376 { #2 }
12377 }
12378 \cs_new:Nn
12379 \@@_latex_paralist_style:nn
12380 {
12381 \str_case:nn
12382 { #1 }
12383 {
12384 { Decimal } { 1 }
12385 { LowerRoman } { i }
12386 { UpperRoman } { I }
12387 { LowerAlpha } { a }
12388 { UpperAlpha } { A }
12389 }
12390 \@@_latex_fancy_list_item_label_delimiter:n
12391 { #2 }

```

```

12392 }
12393 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

12394 ulBeginTight = {%
12395 \group_begin:
12396 \pltopsep=\topsep
12397 \plpartopsep=\partopsep
12398 \begin{compactitem}
12399 },
12400 ulEndTight = {
12401 \end{compactitem}
12402 \group_end:
12403 },
12404 fancyOlBegin = {
12405 \group_begin:
12406 \tl_set:Nn
12407 \l_@@_latex_fancy_list_item_label_number_style_tl
12408 { #1 }
12409 \tl_set:Nn
12410 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12411 { #2 }
12412 \@@_if_option:nTF
12413 { startNumber }
12414 {
12415 \tl_set:Nn
12416 \l_tmpa_tl
12417 { \begin{enumerate} }
12418 }
12419 {
12420 \tl_set:Nn
12421 \l_tmpa_tl
12422 { \begin{enumerate}[] }
12423 \tl_put_right:Nx
12424 \l_tmpa_tl
12425 { \@_latex_paralist_style:nn { #1 } { #2 } }
12426 \tl_put_right:Nn
12427 \l_tmpa_tl
12428 {] }
12429 }
12430 \tl_use:N
12431 \l_tmpa_tl
12432 },
12433 fancyOlEnd = {
12434 \end{enumerate}
12435 \group_end:

```

12436 },

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```
12437 olBeginTight = {%
12438 \group_begin:
12439 \plpartopsep=\partopsep
12440 \pltopsep=\topsep
12441 \begin{compactenum}
12442 },
12443 olEndTight = {
12444 \end{compactenum}
12445 \group_end:
12446 },
12447 fancyOlBeginTight = {
12448 \group_begin:
12449 \tl_set:Nn
12450 \l_@@_latex_fancy_list_item_label_number_style_tl
12451 { #1 }
12452 \tl_set:Nn
12453 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12454 { #2 }
12455 \tl_set:Nn
12456 \l_tmpa_tl
12457 {
12458 \plpartopsep=\partopsep
12459 \pltopsep=\topsep
12460 }
12461 \@@_if_option:nTF
12462 { startNumber }
12463 {
12464 \tl_put_right:Nn
12465 \l_tmpa_tl
12466 { \begin{compactenum} }
12467 }
12468 {
12469 \tl_put_right:Nn
12470 \l_tmpa_tl
12471 { \begin{compactenum}[] }
12472 \tl_put_right:Nx
12473 \l_tmpa_tl
12474 { \@@_latex_paralist_style:nn { #1 } { #2 } }
12475 \tl_put_right:Nn
12476 \l_tmpa_tl
12477 {] }
12478 }
12479 \tl_use:N
```



```

12480 \l_tmpa_tl
12481 },
12482 fancyOlEndTight = {
12483 \end{compactenum}
12484 \group_end:
12485 },
12486 fancyOlItemWithNumber = {
12487 \item
12488 [
12489 \@@_latex_fancy_list_item_label:VVn
12490 \l_@@_latex_fancy_list_item_label_number_style_tl
12491 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12492 { #1 }
12493]
12494 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

12495 dlBeginTight = {
12496 \group_begin:
12497 \plpartopsep=\partopsep
12498 \pltopsep=\topsep
12499 \begin{compactdesc}
12500 },
12501 dlEndTight = {
12502 \end{compactdesc}
12503 \group_end:
12504 }}}
12505 \cs_generate_variant:Nn
12506 \@@_latex_fancy_list_item_label:nnn
12507 { VVn }
12508 }{
12509 \markdownSetup{rendererPrototypes={
12510 ulBeginTight = {\markdownRendererUlBegin},
12511 ulEndTight = {\markdownRendererUlEnd},
12512 fancyOlBegin = {\markdownRendererOlBegin},
12513 fancyOlEnd = {\markdownRendererOlEnd},
12514 olBeginTight = {\markdownRendererOlBegin},
12515 olEndTight = {\markdownRendererOlEnd},
12516 fancyOlBeginTight = {\markdownRendererOlBegin},
12517 fancyOlEndTight = {\markdownRendererOlEnd},
12518 dlBeginTight = {\markdownRendererDlBegin},
12519 dlEndTight = {\markdownRendererDlEnd}}}
12520 }
12521 \ExplSyntaxOff
12522 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

12523 \@ifpackageloaded{unicode-math}{
12524 \markdownSetup{rendererPrototypes={
12525 untickedBox = {\mdlgwhtsquare},
12526 }}
12527 }{
12528 \RequirePackage{amssymb}
12529 \markdownSetup{rendererPrototypes={
12530 untickedBox = {\square},
12531 }}
12532 }
12533 \RequirePackage{csvsimple}
12534 \RequirePackage{fancyvrb}
12535 \RequirePackage{graphicx}
12536 \markdownSetup{rendererPrototypes={
12537 hardLineBreak = {\},
12538 leftBrace = {\textbraceleft},
12539 rightBrace = {\textbraceright},
12540 dollarSign = {\textdollar},
12541 underscore = {\textunderscore},
12542 circumflex = {\textasciicircum},
12543 backslash = {\textbackslash},
12544 tilde = {\textasciitilde},
12545 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

12546 codeSpan = {%
12547 \ifmmode
12548 \text{#1}%
12549 \else
12550 \texttt{#1}%
12551 \fi
12552 }}
12553 \ExplSyntaxOn
12554 \markdownSetup{
12555 rendererPrototypes = {
12556 contentBlock = {

```

---

<sup>34</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

12557 \str_case:nnF
12558 { #1 }
12559 {
12560 { csv }
12561 {
12562 \begin{table}
12563 \begin{center}
12564 \csvautotabular{#3}
12565 \end{center}
12566 \tl_if_empty:nF
12567 { #4 }
12568 { \caption{#4} }
12569 \end{table}
12570 }
12571 { tex } { \markdownEscape{#3} }
12572 }
12573 { \markdownInput{#3} }
12574 },
12575 },
12576 }
12577 \ExplSyntaxOff
12578 \markdownSetup{rendererPrototypes={
12579 image = {%
12580 \begin{figure}%
12581 \begin{center}%
12582 \includegraphics[alt={#1}]{#3}%
12583 \end{center}%
12584 \ifx\empty#4\empty\else
12585 \caption{#4}%
12586 \fi
12587 \end{figure}},
12588 ulBegin = {\begin{itemize}},
12589 ulEnd = {\end{itemize}},
12590 olBegin = {\begin{enumerate}},
12591 olItem = {\item{}},
12592 olItemWithNumber = {\item[#1.]},
12593 olEnd = {\end{enumerate}},
12594 dlBegin = {\begin{description}},
12595 dlItem = {\item[#1]},
12596 dlEnd = {\end{description}},
12597 emphasis = {\emph{#1}},
12598 tickedBox = {\\boxtimes},
12599 halfTickedBox = {\\boxdot}}

```

If identifier attributes appear at the beginning of a section, we make them produce the `\label` macro.

```
12600 \ExplSyntaxOn
```

```

12601 \seq_new:N \l_@@_header_identifiers_seq
12602 \markdownSetup{
12603 rendererPrototypes = {
12604 headerAttributeContextBegin = {
12605 \seq_clear:N \l_@@_header_identifiers_seq
12606 \markdownSetup
12607 {
12608 renderers = {
12609 attributeIdentifier = {
12610 \seq_put_right:Nn
12611 \l_@@_header_identifiers_seq
12612 { ##1 }
12613 },
12614 },
12615 }
12616 },
12617 headerAttributeContextEnd = {
12618 \seq_map_inline:Nn
12619 \l_@@_header_identifiers_seq
12620 { \label { ##1 } }
12621 },
12622 },
12623 }
12624 \ExplSyntaxOff
12625 \markdownSetup{rendererPrototypes={
12626 superscript = {#1},
12627 subscript = {\textsubscript{#1}},
12628 blockQuoteBegin = {\begin{quotation}},
12629 blockQuoteEnd = {\end{quotation}},
12630 inputVerbatim = {\VerbatimInput{#1}},
12631 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
12632 note = {\footnote{#1}}}}

```

### 3.3.5.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

12633 \RequirePackage{ltxcmds}
12634 \ExplSyntaxOn
12635 \cs_gset:Npn
12636 \markdownRendererInputFencedCodePrototype#1#2#3
12637 {
12638 \tl_if_empty:nTF
12639 { #2 }
12640 { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written.

```

12641 {

```

```

12642 \regex_extract_once:nnN
12643 { \w* }
12644 { #2 }
12645 \l_tmpa_seq
12646 \seq_pop_left:NN
12647 \l_tmpa_seq
12648 \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

12649 \ltx@ifpackageloaded
12650 { minted }
12651 {
12652 \catcode`\#=6\relax
12653 \exp_args:NV
12654 \inputminted
12655 \l_tmpa_tl
12656 { #1 }
12657 \catcode`\#=12\relax
12658 }
12659 {

```

When the listings package is loaded, use it for syntax highlighting.

```

12660 \ltx@ifpackageloaded
12661 { listings }
12662 { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostrng were given.

```

12663 { \markdownRendererInputFencedCode{#1}{}} }
12664 }
12665 }
12666 }
12667 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

12668 \ExplSyntaxOn
12669 \def\markdownLATEXStrongEmphasis#1{%
12670 \str_if_in:NnTF
12671 \f@series
12672 { b }
12673 { \textnormal{#1} }
12674 { \textbf{#1} }
12675 }
12676 \ExplSyntaxOff
12677 \markdownSetup{rendererPrototypes={strongEmphasis={%
12678 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support  $\LaTeX$  document classes that do not provide chapters.

```

12679 \@ifundefined{chapter}{%

```

```

12680 \markdownSetup{rendererPrototypes = {
12681 headingOne = {\section{#1}},
12682 headingTwo = {\subsection{#1}},
12683 headingThree = {\subsubsection{#1}},
12684 headingFour = {\paragraph{#1}},
12685 headingFive = {\subparagraph{#1}}}}
12686 }{%
12687 \markdownSetup{rendererPrototypes = {
12688 headingOne = {\chapter{#1}},
12689 headingTwo = {\section{#1}},
12690 headingThree = {\subsection{#1}},
12691 headingFour = {\subsubsection{#1}},
12692 headingFive = {\paragraph{#1}},
12693 headingSix = {\subparagraph{#1}}}}
12694 }%

```

### 3.3.5.2 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

12695 \markdownSetup{
12696 rendererPrototypes = {
12697 ulItem = {%
12698 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
12699 },
12700 },
12701 }
12702 \def\markdownLaTeXUItem{%
12703 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
12704 \item[\markdownLaTeXCheckbox]%
12705 \expandafter\@gobble
12706 \else
12707 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
12708 \item[\markdownLaTeXCheckbox]%
12709 \expandafter\expandafter\expandafter\@gobble
12710 \else
12711 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
12712 \item[\markdownLaTeXCheckbox]%
12713 \expandafter\expandafter\expandafter\expandafter
12714 \expandafter\expandafter\expandafter\@gobble
12715 \else
12716 \item{}%
12717 \fi
12718 \fi
12719 \fi
12720 }

```

### 3.3.5.3 HTML elements

If the `html` option is enabled and we are using  $\text{T}\text{E}\text{X}4\text{ht}$ <sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```
12721 \@ifundefined{HCode}{-}{-}{
12722 \markdownSetup{
12723 rendererPrototypes = {
12724 inlineHtmlTag = {%
12725 \ifvmode
12726 \IgnorePar
12727 \EndP
12728 \fi
12729 \HCode{#1}%
12730 },
12731 inputBlockHtmlElement = {%
12732 \ifvmode
12733 \IgnorePar
12734 \fi
12735 \EndP
12736 \special{t4ht* <#1}%
12737 \par
12738 \ShowPar
12739 },
12740 },
12741 }
12742 }
```

### 3.3.5.4 Citations

Here is a basic implementation for citations that uses the  $\text{L}\text{A}\text{T}\text{E}\text{X}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `BibL $\text{A}\text{T}\text{E}\text{X}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
12743 \newcount\markdownLaTeXCitationsCounter
12744
12745 % Basic implementation
12746 \RequirePackage{gobble}
12747 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
12748 \advance\markdownLaTeXCitationsCounter by 1\relax
12749 \ifx\relax#4\relax
12750 \ifx\relax#5\relax
12751 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12752 \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
12753 \expandafter\expandafter\expandafter
12754 \expandafter\expandafter\expandafter\expandafter
12755 \@gobblethree
```

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

12756 \fi
12757 \else% Before a postnote (#5), dump the accumulator
12758 \ifx\relax#1\relax\else
12759 \cite{#1}%
12760 \fi
12761 \cite[#5]{#6}%
12762 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12763 \else
12764 \expandafter\expandafter\expandafter
12765 \expandafter\expandafter\expandafter\expandafter
12766 \expandafter\expandafter\expandafter
12767 \expandafter\expandafter\expandafter\expandafter
12768 \markdownLaTeXBasicCitations
12769 \fi
12770 \expandafter\expandafter\expandafter
12771 \expandafter\expandafter\expandafter\expandafter{%
12772 \expandafter\expandafter\expandafter
12773 \expandafter\expandafter\expandafter\expandafter}%
12774 \expandafter\expandafter\expandafter
12775 \expandafter\expandafter\expandafter\expandafter{%
12776 \expandafter\expandafter\expandafter
12777 \expandafter\expandafter\expandafter\expandafter}%
12778 \expandafter\expandafter\expandafter
12779 \@gobblethree
12780 \fi
12781 \else% Before a prenote (#4), dump the accumulator
12782 \ifx\relax#1\relax\else
12783 \cite{#1}%
12784 \fi
12785 \ifnum\markdownLaTeXCitationsCounter>1\relax
12786 \space % Insert a space before the prenote in later citations
12787 \fi
12788 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
12789 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12790 \else
12791 \expandafter\expandafter\expandafter
12792 \expandafter\expandafter\expandafter\expandafter
12793 \markdownLaTeXBasicCitations
12794 \fi
12795 \expandafter\expandafter\expandafter{%
12796 \expandafter\expandafter\expandafter}%
12797 \expandafter\expandafter\expandafter{%
12798 \expandafter\expandafter\expandafter}%
12799 \expandafter
12800 \@gobblethree
12801 \fi\markdownLaTeXBasicCitations{#1#2#6},}
12802 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

```



```

12803
12804 % Natbib implementation
12805 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
12806 \advance\markdownLaTeXCitationsCounter by 1\relax
12807 \ifx\relax#3\relax
12808 \ifx\relax#4\relax
12809 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12810 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
12811 \expandafter\expandafter\expandafter
12812 \expandafter\expandafter\expandafter\expandafter
12813 \@gobbletwo
12814 \fi
12815 \else% Before a postnote (#4), dump the accumulator
12816 \ifx\relax#1\relax\else
12817 \citep{#1}%
12818 \fi
12819 \citep[] [#4]{#5}%
12820 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12821 \else
12822 \expandafter\expandafter\expandafter
12823 \expandafter\expandafter\expandafter\expandafter
12824 \expandafter\expandafter\expandafter
12825 \expandafter\expandafter\expandafter\expandafter
12826 \markdownLaTeXNatbibCitations
12827 \fi
12828 \expandafter\expandafter\expandafter
12829 \expandafter\expandafter\expandafter\expandafter{%
12830 \expandafter\expandafter\expandafter
12831 \expandafter\expandafter\expandafter\expandafter}%
12832 \expandafter\expandafter\expandafter
12833 \@gobbletwo
12834 \fi
12835 \else% Before a prenote (#3), dump the accumulator
12836 \ifx\relax#1\relax\relax\else
12837 \citep{#1}%
12838 \fi
12839 \citep[#3] [#4]{#5}%
12840 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12841 \else
12842 \expandafter\expandafter\expandafter
12843 \expandafter\expandafter\expandafter\expandafter
12844 \markdownLaTeXNatbibCitations
12845 \fi
12846 \expandafter\expandafter\expandafter{%
12847 \expandafter\expandafter\expandafter}%
12848 \expandafter
12849 \@gobbletwo

```

```

12850 \fi\markdownLaTeXNatbibCitations{#1,#5}}
12851 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
12852 \advance\markdownLaTeXCitationsCounter by 1\relax
12853 \ifx\relax#3\relax
12854 \ifx\relax#4\relax
12855 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12856 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
12857 \expandafter\expandafter\expandafter
12858 \expandafter\expandafter\expandafter\expandafter
12859 \@gobbletwo
12860 \fi
12861 \else% After a prenote or a postnote, dump the accumulator
12862 \ifx\relax#1\relax\else
12863 \citet{#1}%
12864 \fi
12865 , \citet[#3][#4]{#5}%
12866 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
12867 ,
12868 \else
12869 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
12870 ,
12871 \fi
12872 \fi
12873 \expandafter\expandafter\expandafter
12874 \expandafter\expandafter\expandafter\expandafter
12875 \markdownLaTeXNatbibTextCitations
12876 \expandafter\expandafter\expandafter
12877 \expandafter\expandafter\expandafter\expandafter{%
12878 \expandafter\expandafter\expandafter
12879 \expandafter\expandafter\expandafter\expandafter}%
12880 \expandafter\expandafter\expandafter
12881 \@gobbletwo
12882 \fi
12883 \else% After a prenote or a postnote, dump the accumulator
12884 \ifx\relax#1\relax\relax\else
12885 \citet{#1}%
12886 \fi
12887 , \citet[#3][#4]{#5}%
12888 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
12889 ,
12890 \else
12891 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
12892 ,
12893 \fi
12894 \fi
12895 \expandafter\expandafter\expandafter
12896 \markdownLaTeXNatbibTextCitations

```

```

12897 \expandafter\expandafter\expandafter{%
12898 \expandafter\expandafter\expandafter}%
12899 \expandafter
12900 \@gobbletwo
12901 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
12902
12903 % BibLaTeX implementation
12904 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
12905 \advance\markdownLaTeXCitationsCounter by 1\relax
12906 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12907 \autocites#1[#3][#4]{#5}%
12908 \expandafter\@gobbletwo
12909 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
12910 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
12911 \advance\markdownLaTeXCitationsCounter by 1\relax
12912 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12913 \textcites#1[#3][#4]{#5}%
12914 \expandafter\@gobbletwo
12915 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
12916
12917 \markdownSetup{rendererPrototypes = {
12918 cite = {%
12919 \markdownLaTeXCitationsCounter=1%
12920 \def\markdownLaTeXCitationsTotal{#1}%
12921 \@ifundefined{autocites}{%
12922 \@ifundefined{citep}{%
12923 \expandafter\expandafter\expandafter
12924 \markdownLaTeXBasicCitations
12925 \expandafter\expandafter\expandafter{%
12926 \expandafter\expandafter\expandafter}%
12927 \expandafter\expandafter\expandafter{%
12928 \expandafter\expandafter\expandafter}%
12929 }{%
12930 \expandafter\expandafter\expandafter
12931 \markdownLaTeXNatbibCitations
12932 \expandafter\expandafter\expandafter{%
12933 \expandafter\expandafter\expandafter}%
12934 }%
12935 }{%
12936 \expandafter\expandafter\expandafter
12937 \markdownLaTeXBibLaTeXCitations
12938 \expandafter{\expandafter}%
12939 }},
12940 textCite = {%
12941 \markdownLaTeXCitationsCounter=1%
12942 \def\markdownLaTeXCitationsTotal{#1}%
12943 \@ifundefined{autocites}{%

```

```

12944 \@ifundefined{citep}{%
12945 \expandafter\expandafter\expandafter
12946 \markdownLaTeXBasicTextCitations
12947 \expandafter\expandafter\expandafter{%
12948 \expandafter\expandafter\expandafter}%
12949 \expandafter\expandafter\expandafter{%
12950 \expandafter\expandafter\expandafter}%
12951 }{%
12952 \expandafter\expandafter\expandafter
12953 \markdownLaTeXNatbibTextCitations
12954 \expandafter\expandafter\expandafter{%
12955 \expandafter\expandafter\expandafter}%
12956 }%
12957 }{%
12958 \expandafter\expandafter\expandafter
12959 \markdownLaTeXBibLaTeXTextCitations
12960 \expandafter{\expandafter}%
12961 }}}}
```

### 3.3.5.5 Links

Here is an implementation for hypertext links and relative references.

```

12962 \RequirePackage{url}
12963 \RequirePackage{expl3}
12964 \ExplSyntaxOn
12965 \def\markdownRendererLinkPrototype#1#2#3#4{
12966 \tl_set:Nn \l_tmpa_tl { #1 }
12967 \tl_set:Nn \l_tmpb_tl { #2 }
12968 \bool_set:Nn
12969 \l_tmpa_bool
12970 {
12971 \tl_if_eq_p:NN
12972 \l_tmpa_tl
12973 \l_tmpb_tl
12974 }
12975 \tl_set:Nn \l_tmpa_tl { #4 }
12976 \bool_set:Nn
12977 \l_tmpb_bool
12978 {
12979 \tl_if_empty_p:N
12980 \l_tmpa_tl
12981 }
12982 \bool_if:nTF
12983 {
```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```

12982 \bool_if:nTF
12983 {
```

```

12984 \l_tmpa_bool && \l_tmpb_bool
12985 }
12986 {
12987 \markdownLaTeXRendererAutolink { #2 } { #3 }
12988 }{
12989 \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
12990 }
12991 }
12992 \def\markdownLaTeXRendererAutolink#1#2{%
If the URL begins with a hash sign, then we assume that it is a relative reference.
Otherwise, we assume that it is an absolute URL.
12993 \tl_set:Nn
12994 \l_tmpa_tl
12995 { #2 }
12996 \tl_trim_spaces:N
12997 \l_tmpa_tl
12998 \tl_set:Nx
12999 \l_tmpb_tl
13000 {
13001 \tl_range:Nnn
13002 \l_tmpa_tl
13003 { 1 }
13004 { 1 }
13005 }
13006 \str_if_eq:NNTF
13007 \l_tmpb_tl
13008 \c_hash_str
13009 {
13010 \tl_set:Nx
13011 \l_tmpb_tl
13012 {
13013 \tl_range:Nnn
13014 \l_tmpa_tl
13015 { 2 }
13016 { -1 }
13017 }
13018 \exp_args:NV
13019 \ref
13020 \l_tmpb_tl
13021 }{
13022 \url { #2 }
13023 }
13024 }
13025 \ExplSyntaxOff
13026 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
13027 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

### 3.3.5.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```
13028 \newcount\markdownLaTeXRowCounter
13029 \newcount\markdownLaTeXRowTotal
13030 \newcount\markdownLaTeXColumnCounter
13031 \newcount\markdownLaTeXColumnTotal
13032 \newtoks\markdownLaTeXTable
13033 \newtoks\markdownLaTeXTableAlignment
13034 \newtoks\markdownLaTeXTableEnd
13035 \AtBeginDocument{%
13036 \@ifpackageloaded{booktabs}{%
13037 \def\markdownLaTeXTopRule{\toprule}%
13038 \def\markdownLaTeXMidRule{\midrule}%
13039 \def\markdownLaTeXBottomRule{\bottomrule}%
13040 }{%
13041 \def\markdownLaTeXTopRule{\hline}%
13042 \def\markdownLaTeXMidRule{\hline}%
13043 \def\markdownLaTeXBottomRule{\hline}%
13044 }%
13045 }
13046 \markdownSetup{rendererPrototypes={
13047 table = {%
13048 \markdownLaTeXTable={}%
13049 \markdownLaTeXTableAlignment={}%
13050 \markdownLaTeXTableEnd={%
13051 \markdownLaTeXBottomRule
13052 \end{tabular}}%
13053 \ifx\empty#1\empty\else
13054 \addto@hook\markdownLaTeXTable{%
13055 \begin{table}
13056 \centering}%
13057 \addto@hook\markdownLaTeXTableEnd{%
13058 \caption{#1}
13059 \end{table}}%
13060 \fi
13061 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
13062 \markdownLaTeXRowCounter=0%
13063 \markdownLaTeXRowTotal=#2%
13064 \markdownLaTeXColumnTotal=#3%
13065 \markdownLaTeXRenderTableRow
13066 }
13067 }}
13068 \def\markdownLaTeXRenderTableRow#1{%
13069 \markdownLaTeXColumnCounter=0%
13070 \ifnum\markdownLaTeXRowCounter=0\relax
13071 \markdownLaTeXReadAlignments#1%
```

```

13072 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
13073 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
13074 \the\markdownLaTeXTableAlignment}}%
13075 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
13076 \else
13077 \markdownLaTeXRenderTableCell#1%
13078 \fi
13079 \ifnum\markdownLaTeXRowCount=1\relax
13080 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
13081 \fi
13082 \advance\markdownLaTeXRowCount by 1\relax
13083 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
13084 \the\markdownLaTeXTable
13085 \the\markdownLaTeXTableEnd
13086 \expandafter\@gobble
13087 \fi\markdownLaTeXRenderTableRow}
13088 \def\markdownLaTeXReadAlignments#1{%
13089 \advance\markdownLaTeXColumnCounter by 1\relax
13090 \if#1d%
13091 \addto@hook\markdownLaTeXTableAlignment{1}%
13092 \else
13093 \addto@hook\markdownLaTeXTableAlignment{#1}%
13094 \fi
13095 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
13096 \expandafter\@gobble
13097 \fi\markdownLaTeXReadAlignments}
13098 \def\markdownLaTeXRenderTableCell#1{%
13099 \advance\markdownLaTeXColumnCounter by 1\relax
13100 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
13101 \addto@hook\markdownLaTeXTable{#1&}%
13102 \else
13103 \addto@hook\markdownLaTeXTable{#1\\}%
13104 \expandafter\@gobble
13105 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.5.7 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

13106
13107 \markdownIfOption{lineBlocks}{%
13108 \RequirePackage{verse}
13109 \markdownSetup{rendererPrototypes={
13110 lineBlockBegin = {%
13111 \begingroup
13112 \def\markdownRendererHardLineBreak{\\}%
13113 \begin{verse}%

```

```

13114 },
13115 lineBlockEnd = {%
13116 \end{verse}%
13117 \endgroup
13118 },
13119 }}
13120 }{}
13121

```

### 3.3.5.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

13122 \ExplSyntaxOn
13123 \keys_define:nn
13124 { markdown/jekyllData }
13125 {
13126 author .code:n = { \author{#1} },
13127 date .code:n = { \date{#1} },
13128 title .code:n = { \title{#1} },
13129 }

```

To complement the default setup of our key–values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```

13130 \markdownSetup{
13131 rendererPrototypes = {
13132 jekyllDataEnd = {
13133 \AddToHook{begindocument/end}{\maketitle}
13134 },
13135 },
13136 }
13137 \ExplSyntaxOff

```

### 3.3.5.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

13138 \markdownIfOption{strikeThrough}{%
13139 \RequirePackage{soulutf8}%
13140 \markdownSetup{
13141 rendererPrototypes = {
13142 strikeThrough = {%
13143 \st{#1}%
13144 },

```



```

13145 }
13146 }
13147 }{}

```

### 3.3.5.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```

13148 \markdownIfOption{mark}{%
13149 \RequirePackage{soulutf8}%
13150 \markdownSetup{
13151 rendererPrototypes = {
13152 mark = {%
13153 \hl{#1}%
13154 },
13155 }
13156 }
13157 }{}

```

### 3.3.5.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values.

```

13158 \ExplSyntaxOn
13159 \@@_if_option:nT
13160 { linkAttributes }
13161 {
13162 \RequirePackage{graphicx}
13163 \markdownSetup{
13164 rendererPrototypes = {
13165 imageAttributeContextBegin = {
13166 \group_begin:
13167 \markdownSetup{
13168 rendererPrototypes = {
13169 attributeKeyValue = {
13170 \setkeys
13171 { Gin }
13172 { { ##1 } = { ##2 } }
13173 },
13174 },
13175 }
13176 },
13177 imageAttributeContextEnd = {
13178 \group_end:
13179 },

```

```

13180 },
13181 }
13182 }
13183 \ExplSyntaxOff

```

### 3.3.5.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

13184 \ExplSyntaxOn
13185 \cs_gset:Npn
13186 \markdownRendererInputRawInlinePrototype#1#2
13187 {
13188 \str_case:nnF
13189 { #2 }
13190 {
13191 { latex }
13192 {
13193 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13194 { #1 }
13195 { tex }
13196 }
13197 }
13198 {
13199 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13200 { #1 }
13201 { #2 }
13202 }
13203 }
13204 \cs_gset:Npn
13205 \markdownRendererInputRawBlockPrototype#1#2
13206 {
13207 \str_case:nnF
13208 { #2 }
13209 {
13210 { latex }
13211 {
13212 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13213 { #1 }
13214 { tex }
13215 }
13216 }
13217 {
13218 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13219 { #1 }
13220 { #2 }
13221 }

```

```

13222 }
13223 \ExplSyntaxOff
13224 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}

```

### 3.3.6 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `inputenc` package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` package.

```

13225 \newcommand\markdownMakeOther{%
13226 \count0=128\relax
13227 \loop
13228 \catcode\count0=11\relax
13229 \advance\count0 by 1\relax
13230 \ifnum\count0<256\repeat}%

```

## 3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the documentation is scarce) the majority of the plain TeX format required by the plain TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` L<sup>A</sup>T<sub>E</sub>X package.

```

13231 \def\markdownMakeOther{%
13232 \count0=128\relax
13233 \loop
13234 \catcode\count0=11\relax
13235 \advance\count0 by 1\relax
13236 \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTeXt.

```

13237 \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```

13238 \long\def\inputmarkdown{%
13239 \dosingleempty

```

```

13240 \doinputmarkdown}%
13241 \long\def\doinputmarkdown[#1]#2{%
13242 \begingroup
13243 \iffirstargument
13244 \setupmarkdown[#1]%
13245 \fi
13246 \markdownInput{#2}%
13247 \endgroup}%

```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's  $\TeX$ , trailing spaces are removed very early on when a line is being put to the input buffer. [12, sec. 31]. According to Eijkhout [13, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua) $\TeX$ , but Con $\TeX$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\TeX$ t MkIV and therefore to insert hard line breaks into markdown text.

```

13248 \startluacode
13249 document.markdown_buffering = false
13250 local function preserve_trailing_spaces(line)
13251 if document.markdown_buffering then
13252 line = line:gsub("[\t][\t]$", "\t\t")
13253 end
13254 return line
13255 end
13256 resolvers.installinputlinehandler(preserve_trailing_spaces)
13257 \stopluacode
13258 \begingroup
13259 \catcode\|=0%
13260 \catcode\|=12%
13261 |gdef|startmarkdown{%
13262 |ctxlua{document.markdown_buffering = true}%
13263 |markdownReadAndConvert{\stopmarkdown}%
13264 |stopmarkdown}%
13265 |gdef|stopmarkdown{%
13266 |ctxlua{document.markdown_buffering = false}%
13267 |markdownEnd}%
13268 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in Con $\TeX$ t themes provided with the Markdown package.

```

13269 \ExplSyntaxOn
13270 \cs_gset:Nn

```

```

13271 \@@_load_theme:nn
13272 {
 Determine whether a file named t-markdowntheme<munged theme name>.tex
 exists. If it does, load it. Otherwise, try loading a plain TEX theme instead.
13273 \file_if_exist:nTF
13274 { t - markdown theme #2.tex }
13275 {
13276 \msg_info:nnn
13277 { markdown }
13278 { loading-context-theme }
13279 { #1 }
13280 \usemodule
13281 [t]
13282 [markdown theme #2]
13283 }
13284 {
13285 \@@_plain_tex_load_theme:nn
13286 { #1 }
13287 { #2 }
13288 }
13289 }
13290 \msg_new:nnn
13291 { markdown }
13292 { loading-context-theme }
13293 { Loading~ConTeXt~Markdown~theme~#1 }
13294 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConT<sub>E</sub>Xt theme provides default definitions for token renderer prototypes. First, the ConT<sub>E</sub>Xt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```
13295 \markdownLoadPlainTeXTheme
```

Next, the ConT<sub>E</sub>Xt theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

13296 \markdownIfOption{plain}{\iffalse}{\iftrue}
13297 \def\markdownRendererHardLineBreakPrototype{\blank}%
13298 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
13299 \def\markdownRendererRightBracePrototype{\textbraceright}%
13300 \def\markdownRendererDollarSignPrototype{\textdollar}%
13301 \def\markdownRendererPercentSignPrototype{\percent}%
13302 \def\markdownRendererUnderscorePrototype{\textunderscore}%
13303 \def\markdownRendererCircumflexPrototype{\textcircumflex}%

```

```

13304 \def\markdownRendererBackslashPrototype{\textbackslash}%
13305 \def\markdownRendererTildePrototype{\textasciitilde}%
13306 \def\markdownRendererPipePrototype{\char`|}%
13307 \def\markdownRendererLinkPrototype#1#2#3#4{%
13308 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
13309 \fi\texttt<\hyphenatedurl{#3}>}}%
13310 \usemodule[database]
13311 \defineseparatedlist
13312 [MarkdownConTeXtCSV]
13313 [separator={,},
13314 before=\bTABLE,after=\eTABLE,
13315 first=\bTR,last=\eTR,
13316 left=\bTD,right=\eTD]
13317 \def\markdownConTeXtCSV{csv}
13318 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13319 \def\markdownConTeXtCSV@arg{#1}%
13320 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
13321 \placetable[] [tab:#1]{#4}{%
13322 \processseparatedfile [MarkdownConTeXtCSV] [#3]}%
13323 \else
13324 \markdownInput{#3}%
13325 \fi}%
13326 \def\markdownRendererImagePrototype#1#2#3#4{%
13327 \placefigure[] []{#4}{\externalfigure[#3]}%
13328 \def\markdownRendererULBeginPrototype{\startitemize}%
13329 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
13330 \def\markdownRendererULItemPrototype{\item}%
13331 \def\markdownRendererULEndPrototype{\stopitemize}%
13332 \def\markdownRendererULEndTightPrototype{\stopitemize}%
13333 \def\markdownRendererOLBeginPrototype{\startitemize[n]}%
13334 \def\markdownRendererOLBeginTightPrototype{\startitemize[packed,n]}%
13335 \def\markdownRendererOLItemPrototype{\item}%
13336 \def\markdownRendererOLItemWithNumberPrototype#1{\sym{#1.}}%
13337 \def\markdownRendererOLEndPrototype{\stopitemize}%
13338 \def\markdownRendererOLEndTightPrototype{\stopitemize}%
13339 \definedescription
13340 [MarkdownConTeXtDlItemPrototype]
13341 [location=hanging,
13342 margin=standard,
13343 headstyle=bold]%
13344 \definestartstop
13345 [MarkdownConTeXtDlPrototype]
13346 [before=\blank,
13347 after=\blank]%
13348 \definestartstop
13349 [MarkdownConTeXtDlTightPrototype]
13350 [before=\blank\startpacked,

```

```

13351 after=\stoppacked\blank]%
13352 \def\markdownRendererDlBeginPrototype{%
13353 \startMarkdownConTeXtDlPrototype}%
13354 \def\markdownRendererDlBeginTightPrototype{%
13355 \startMarkdownConTeXtDlTightPrototype}%
13356 \def\markdownRendererDlItemPrototype#1{%
13357 \startMarkdownConTeXtDlItemPrototype{#1}}%
13358 \def\markdownRendererDlItemEndPrototype{%
13359 \stopMarkdownConTeXtDlItemPrototype}%
13360 \def\markdownRendererDlEndPrototype{%
13361 \stopMarkdownConTeXtDlPrototype}%
13362 \def\markdownRendererDlEndTightPrototype{%
13363 \stopMarkdownConTeXtDlTightPrototype}%
13364 \def\markdownRendererEmphasisPrototype#1{\em#1}%
13365 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13366 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
13367 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
13368 \def\markdownRendererLineBlockBeginPrototype{%
13369 \begingroup
13370 \def\markdownRendererHardLineBreak{
13371 }%
13372 \startlines
13373 }%
13374 \def\markdownRendererLineBlockEndPrototype{%
13375 \stoplines
13376 \endgroup
13377 }%
13378 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13379 \ExplSyntaxOn
13380 \cs_gset:Npn
13381 \markdownRendererInputFencedCodePrototype#1#2#3
13382 {
13383 \tl_if_empty:nTF
13384 { #2 }
13385 { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptying [latex] [option=TEX]

```

```

\starttext
 \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
 \stopmarkdown
\stoptext

```

```

13386 {
13387 \regex_extract_once:nnN
13388 { \w* }
13389 { #2 }
13390 \l_tmpa_seq
13391 \seq_pop_left:NN
13392 \l_tmpa_seq
13393 \l_tmpa_tl
13394 \typefile[\l_tmpa_tl] []{#1}
13395 }
13396 }
13397 \ExplSyntaxOff
13398 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
13399 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
13400 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
13401 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
13402 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
13403 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
13404 \def\markdownRendererThematicBreakPrototype{%
13405 \blackrule[height=1pt, width=\hsize]}%
13406 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
13407 \def\markdownRendererTickedBoxPrototype{\boxtimes$}
13408 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
13409 \def\markdownRendererUntickedBoxPrototype{\square$}
13410 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
13411 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
13412 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
13413 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

13414 \newcount\markdownConTeXtRowCounter
13415 \newcount\markdownConTeXtRowTotal

```



```

13416 \newcount\markdownConTeXtColumnCounter
13417 \newcount\markdownConTeXtColumnTotal
13418 \newtoks\markdownConTeXtTable
13419 \newtoks\markdownConTeXtTableFloat
13420 \def\markdownRendererTablePrototype#1#2#3{%
13421 \markdownConTeXtTable={}%
13422 \ifx\empty#1\empty
13423 \markdownConTeXtTableFloat={%
13424 \the\markdownConTeXtTable}%
13425 \else
13426 \markdownConTeXtTableFloat={%
13427 \placetable{#1}{\the\markdownConTeXtTable}}%
13428 \fi
13429 \begingroup
13430 \setupTABLE[r][each][topframe=off,bottomframe=off,leftframe=off,rightframe=off]
13431 \setupTABLE[c][each][topframe=off,bottomframe=off,leftframe=off,rightframe=off]
13432 \setupTABLE[r][1][topframe=on,bottomframe=on]
13433 \setupTABLE[r][#1][bottomframe=on]
13434 \markdownConTeXtRowCounter=0%
13435 \markdownConTeXtRowTotal=#2%
13436 \markdownConTeXtColumnTotal=#3%
13437 \markdownConTeXtRenderTableRow}
13438 \def\markdownConTeXtRenderTableRow#1{%
13439 \markdownConTeXtColumnCounter=0%
13440 \ifnum\markdownConTeXtRowCounter=0\relax
13441 \markdownConTeXtReadAlignments#1%
13442 \markdownConTeXtTable={\bTABLE}%
13443 \else
13444 \markdownConTeXtTable=\expandafter{%
13445 \the\markdownConTeXtTable\bTR}%
13446 \markdownConTeXtRenderTableCell#1%
13447 \markdownConTeXtTable=\expandafter{%
13448 \the\markdownConTeXtTable\eTR}%
13449 \fi
13450 \advance\markdownConTeXtRowCounter by 1\relax
13451 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
13452 \markdownConTeXtTable=\expandafter{%
13453 \the\markdownConTeXtTable\eTABLE}%
13454 \the\markdownConTeXtTableFloat
13455 \endgroup
13456 \expandafter\gobbleoneargument
13457 \fi\markdownConTeXtRenderTableRow}
13458 \def\markdownConTeXtReadAlignments#1{%
13459 \advance\markdownConTeXtColumnCounter by 1\relax
13460 \if#1d%
13461 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13462 \fi\if#1l%

```

```

13463 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13464 \fi\if#1c%
13465 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
13466 \fi\if#1r%
13467 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
13468 \fi
13469 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13470 \expandafter\gobbleoneargument
13471 \fi\markdownConTeXtReadAlignments}
13472 \def\markdownConTeXtRenderTableCell#1{%
13473 \advance\markdownConTeXtColumnCounter by 1\relax
13474 \markdownConTeXtTable=\expandafter{%
13475 \the\markdownConTeXtTable\bTD#1\eTD}%
13476 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13477 \expandafter\gobbleoneargument
13478 \fi\markdownConTeXtRenderTableCell}

```

### 3.4.3.3 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```

13479 \ExplSyntaxOn
13480 \cs_gset:Npn
13481 \markdownRendererInputRawInlinePrototype#1#2
13482 {
13483 \str_case:nnF
13484 { #2 }
13485 {
13486 { latex }
13487 {
13488 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13489 { #1 }
13490 { context }
13491 }
13492 }
13493 {
13494 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13495 { #1 }
13496 { #2 }
13497 }
13498 }
13499 \cs_gset:Npn
13500 \markdownRendererInputRawBlockPrototype#1#2
13501 {
13502 \str_case:nnF
13503 { #2 }
13504 {

```

```

13505 { context }
13506 {
13507 \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13508 { #1 }
13509 { tex }
13510 }
13511 }
13512 {
13513 \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13514 { #1 }
13515 { #2 }
13516 }
13517 }
13518 \cs_gset_eq:NN
13519 \markdownRendererInputRawBlockPrototype
13520 \markdownRendererInputRawInlinePrototype
13521 \fi % Closes ``\markdownIfOption{plain}{\iffalse}{\iftrue}`
13522 \ExplSyntaxOff
13523 \stopmodule
13524 \protect

```

At the end of the ConTeXt module, we load the `witiko/markdown/defaults` ConTeXt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

13525 \markdownIfOption{noDefaults}{}{
13526 \setupmarkdown[theme=witiko/markdown/defaults]
13527 }
13528 \stopmodule
13529 \protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).

- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [8] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [9] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [10] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [11] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [12] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [13] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician’s Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

|                       |                                   |
|-----------------------|-----------------------------------|
| autoIdentifiers       | 18, 30, 71, 85                    |
| blankBeforeBlockquote | 18                                |
| blankBeforeCodeFence  | 19                                |
| blankBeforeDivFence   | 19                                |
| blankBeforeHeading    | 19                                |
| blankBeforeList       | 20                                |
| bracketedSpans        | 20, 73                            |
| breakableBlockquotes  | 20                                |
| cacheDir              | 4, 15, 17, 53, 127, 276, 344, 356 |
| citationNbsps         | 21                                |
| citations             | 21, 76                            |
| codeSpans             | 22                                |

|                                   |                         |
|-----------------------------------|-------------------------|
| contentBlocks                     | 17, 22                  |
| contentBlocksLanguageMap          | 17                      |
| contentLevel                      | 23                      |
| debugExtensions                   | 9, 17, 23, 273          |
| debugExtensionsFileName           | 17, 23                  |
| defaultOptions                    | 9, 47, 324              |
| definitionLists                   | 23, 80                  |
| eagerCache                        | 15                      |
| entities.char_entity              | 185                     |
| entities.dec_entity               | 185                     |
| entities.hex_entity               | 185                     |
| entities.hex_entity_with_x_char   | 185                     |
| expandtabs                        | 238                     |
| expectJekyllData                  | 24                      |
| extensions                        | 25, 134, 279            |
| extensions.bracketed_spans        | 279                     |
| extensions.citations              | 280                     |
| extensions.content_blocks         | 284                     |
| extensions.definition_lists       | 287                     |
| extensions.fancy_lists            | 289                     |
| extensions.fenced_code            | 295                     |
| extensions.fenced_divs            | 299                     |
| extensions.header_attributes      | 303                     |
| extensions.inline_code_attributes | 305                     |
| extensions.jekyll_data            | 321                     |
| extensions.line_blocks            | 305                     |
| extensions.link_attributes        | 307                     |
| extensions.mark                   | 306                     |
| extensions.notes                  | 308                     |
| extensions.pipe_table             | 310                     |
| extensions.raw_inline             | 315                     |
| extensions.strike_through         | 315                     |
| extensions.subscripts             | 316                     |
| extensions.superscripts           | 317                     |
| extensions.tex_math               | 317                     |
| fancyLists                        | 27, 94–99, 357          |
| fencedCode                        | 27, 35, 77, 83, 99, 355 |
| fencedCodeAttributes              | 28, 71                  |
| fencedDiv                         | 84                      |
| fencedDivs                        | 28, 37                  |

|                                              |                                                 |
|----------------------------------------------|-------------------------------------------------|
| finalizeCache                                | 16, 18, 29, 29, 53, 126, 278                    |
| frozenCache                                  | 18, 29, 53, 126, 129, 130, 355, 356             |
| frozenCacheCounter                           | 29, 278, 348, 349                               |
| frozenCacheFileName                          | 18, 29, 53, 278                                 |
| gfmAutoIdentifiers                           | 18, 29, 71, 85                                  |
| hashEnumerators                              | 30                                              |
| headerAttributes                             | 30, 37, 71, 85                                  |
| html                                         | 31, 87, 88, 367                                 |
| hybrid                                       | 31, 36, 42, 44, 56, 64, 101, 127, 190, 239, 348 |
| inlineCodeAttributes                         | 31, 71, 78                                      |
| inlineNotes                                  | 32                                              |
| \input                                       | 50                                              |
| \inputmarkdown                               | 131, 132, 379                                   |
| inputTempFileName                            | 54, 56, 345, 346                                |
| iterlines                                    | 238                                             |
| jekyllData                                   | 3, 24, 25, 32, 107–111                          |
| languages_json                               | 284, 284                                        |
| lineBlocks                                   | 33, 90                                          |
| linkAttributes                               | 33, 71, 89, 92, 256, 377                        |
| mark                                         | 34, 92, 377                                     |
| \markdown                                    | 125                                             |
| markdown                                     | 124, 125, 350, 351                              |
| markdown*                                    | 124, 125, 126, 350                              |
| \markdown_jekyll_data_concatenate_address:NN | 339                                             |
| \markdown_jekyll_data_pop:                   | 340                                             |
| \markdown_jekyll_data_push:nN                | 340                                             |
| \markdown_jekyll_data_push_address_segment:n | 338                                             |
| \markdown_jekyll_data_set_keyval:Nn          | 340                                             |
| \markdown_jekyll_data_set_keyvals:nn         | 340                                             |
| \markdown_jekyll_data_update_address_tls:    | 339                                             |
| \markdownBegin                               | 49, 49, 50, 123, 125, 132                       |
| \markdownCleanup                             | 344                                             |
| \markdownEnd                                 | 49, 49, 50, 123, 125, 132                       |
| \markdownError                               | 123, 123                                        |
| \markdownEscape                              | 49, 51, 349                                     |
| \markdownIfOption                            | 52                                              |
| \markdownIfSnippetExists                     | 66                                              |

|                                                                  |                                |
|------------------------------------------------------------------|--------------------------------|
| <code>\markdownInfo</code>                                       | 123, 123                       |
| <code>\markdownInput</code>                                      | 49, 50, 124–126, 132, 348, 350 |
| <code>\markdownInputFileStream</code>                            | 344                            |
| <code>\markdownInputPlainTeX</code>                              | 350                            |
| <code>\markdownLoadPlainTeXTheme</code>                          | 127, 134, 333                  |
| <code>\markdownLuaExecute</code>                                 | 347, 348                       |
| <code>\markdownLuaOptions</code>                                 | 342, 344                       |
| <code>\markdownMakeOther</code>                                  | 123, 379                       |
| <code>\markdownOptionFinalizeCache</code>                        | 53                             |
| <code>\markdownOptionFrozenCache</code>                          | 53                             |
| <code>\markdownOptionHybrid</code>                               | 56                             |
| <code>\markdownOptionInputTempFileName</code>                    | 53                             |
| <code>\markdownOptionNoDefaults</code>                           | 55                             |
| <code>\markdownOptionOutputDir</code>                            | 53, 54                         |
| <code>\markdownOptionPlain</code>                                | 55                             |
| <code>\markdownOptionStripPercentSigns</code>                    | 56                             |
| <code>\markdownOutputFileStream</code>                           | 344                            |
| <code>\markdownPrepare</code>                                    | 344                            |
| <code>\markdownPrepareLuaOptions</code>                          | 342                            |
| <code>\markdownReadAndConvert</code>                             | 123, 345, 350, 351, 380        |
| <code>\markdownReadAndConvertProcessLine</code>                  | 346, 347                       |
| <code>\markdownReadAndConvertStripPercentSigns</code>            | 345                            |
| <code>\markdownReadAndConvertTab</code>                          | 344                            |
| <code>\markdownRendererAttributeClassName</code>                 | 72                             |
| <code>\markdownRendererAttributeIdentifier</code>                | 71                             |
| <code>\markdownRendererAttributeKeyValue</code>                  | 72                             |
| <code>\markdownRendererBlockQuoteBegin</code>                    | 73                             |
| <code>\markdownRendererBlockQuoteEnd</code>                      | 73                             |
| <code>\markdownRendererBracketedSpanAttributeContextBegin</code> | 73                             |
| <code>\markdownRendererBracketedSpanAttributeContextEnd</code>   | 73                             |
| <code>\markdownRendererCite</code>                               | 76, 76                         |
| <code>\markdownRendererCodeSpan</code>                           | 77                             |
| <code>\markdownRendererCodeSpanAttributeContextBegin</code>      | 78                             |
| <code>\markdownRendererCodeSpanAttributeContextEnd</code>        | 78                             |
| <code>\markdownRendererContentBlock</code>                       | 78, 79                         |
| <code>\markdownRendererContentBlockCode</code>                   | 79                             |
| <code>\markdownRendererContentBlockOnlineImage</code>            | 79                             |
| <code>\markdownRendererDisplayMath</code>                        | 106                            |
| <code>\markdownRendererDlBegin</code>                            | 80                             |
| <code>\markdownRendererDlBeginTight</code>                       | 80                             |
| <code>\markdownRendererDlDefinitionBegin</code>                  | 81                             |
| <code>\markdownRendererDlDefinitionEnd</code>                    | 81                             |

|                                                               |         |
|---------------------------------------------------------------|---------|
| <code>\markdownRendererDlEnd</code>                           | 82      |
| <code>\markdownRendererDlEndTight</code>                      | 82      |
| <code>\markdownRendererDlItem</code>                          | 80      |
| <code>\markdownRendererDlItemEnd</code>                       | 81      |
| <code>\markdownRendererDocumentBegin</code>                   | 93      |
| <code>\markdownRendererDocumentEnd</code>                     | 93      |
| <code>\markdownRendererEllipsis</code>                        | 38, 82  |
| <code>\markdownRendererEmphasis</code>                        | 83, 113 |
| <code>\markdownRendererFancyOlBegin</code>                    | 95, 95  |
| <code>\markdownRendererFancyOlBeginTight</code>               | 95      |
| <code>\markdownRendererFancyOlEnd</code>                      | 98      |
| <code>\markdownRendererFancyOlEndTight</code>                 | 98      |
| <code>\markdownRendererFancyOlItem</code>                     | 96      |
| <code>\markdownRendererFancyOlItemEnd</code>                  | 97      |
| <code>\markdownRendererFancyOlItemWithNumber</code>           | 97      |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code> | 83      |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code>   | 83      |
| <code>\markdownRendererFencedDivAttributeContextBegin</code>  | 84      |
| <code>\markdownRendererFencedDivAttributeContextEnd</code>    | 84      |
| <code>\markdownRendererHalfTickedBox</code>                   | 107     |
| <code>\markdownRendererHardLineBreak</code>                   | 91      |
| <code>\markdownRendererHeaderAttributeContextBegin</code>     | 85      |
| <code>\markdownRendererHeaderAttributeContextEnd</code>       | 85      |
| <code>\markdownRendererHeadingFive</code>                     | 86      |
| <code>\markdownRendererHeadingFour</code>                     | 86      |
| <code>\markdownRendererHeadingOne</code>                      | 85      |
| <code>\markdownRendererHeadingSix</code>                      | 87      |
| <code>\markdownRendererHeadingThree</code>                    | 86      |
| <code>\markdownRendererHeadingTwo</code>                      | 86      |
| <code>\markdownRendererImage</code>                           | 88      |
| <code>\markdownRendererImageAttributeContextBegin</code>      | 89      |
| <code>\markdownRendererImageAttributeContextEnd</code>        | 89      |
| <code>\markdownRendererInlineHtmlComment</code>               | 87      |
| <code>\markdownRendererInlineHtmlTag</code>                   | 87      |
| <code>\markdownRendererInlineMath</code>                      | 106     |
| <code>\markdownRendererInputBlockHtmlElement</code>           | 88      |
| <code>\markdownRendererInputFencedCode</code>                 | 77      |
| <code>\markdownRendererInputRawBlock</code>                   | 99      |
| <code>\markdownRendererInputRawInline</code>                  | 99      |
| <code>\markdownRendererInputVerbatim</code>                   | 77      |
| <code>\markdownRendererInterblockSeparator</code>             | 89      |
| <code>\markdownRendererJekyllDataBegin</code>                 | 107     |



|                                                          |         |
|----------------------------------------------------------|---------|
| <code>\markdownRendererJekyllDataBoolean</code>          | 109     |
| <code>\markdownRendererJekyllDataEmpty</code>            | 111     |
| <code>\markdownRendererJekyllDataEnd</code>              | 108     |
| <code>\markdownRendererJekyllDataMappingBegin</code>     | 108     |
| <code>\markdownRendererJekyllDataMappingEnd</code>       | 108     |
| <code>\markdownRendererJekyllDataNumber</code>           | 110     |
| <code>\markdownRendererJekyllDataSequenceBegin</code>    | 109     |
| <code>\markdownRendererJekyllDataSequenceEnd</code>      | 109     |
| <code>\markdownRendererJekyllDataString</code>           | 110     |
| <code>\markdownRendererLineBlockBegin</code>             | 90      |
| <code>\markdownRendererLineBlockEnd</code>               | 90      |
| <code>\markdownRendererLink</code>                       | 91, 113 |
| <code>\markdownRendererLinkAttributeContextBegin</code>  | 92      |
| <code>\markdownRendererLinkAttributeContextEnd</code>    | 92      |
| <code>\markdownRendererMark</code>                       | 92      |
| <code>\markdownRendererNbsp</code>                       | 93      |
| <code>\markdownRendererNote</code>                       | 94      |
| <code>\markdownRendererOlBegin</code>                    | 94      |
| <code>\markdownRendererOlBeginTight</code>               | 94      |
| <code>\markdownRendererOlEnd</code>                      | 97      |
| <code>\markdownRendererOlEndTight</code>                 | 98      |
| <code>\markdownRendererOlItem</code>                     | 38, 95  |
| <code>\markdownRendererOlItemEnd</code>                  | 96      |
| <code>\markdownRendererOlItemWithNumber</code>           | 38, 96  |
| <code>\markdownRendererParagraphSeparator</code>         | 90      |
| <code>\markdownRendererReplacementCharacter</code>       | 100     |
| <code>\markdownRendererSectionBegin</code>               | 100     |
| <code>\markdownRendererSectionEnd</code>                 | 100     |
| <code>\markdownRendererSoftLineBreak</code>              | 91      |
| <code>\markdownRendererStrikeThrough</code>              | 103     |
| <code>\markdownRendererStrongEmphasis</code>             | 83      |
| <code>\markdownRendererSubscript</code>                  | 104     |
| <code>\markdownRendererSuperscript</code>                | 104     |
| <code>\markdownRendererTable</code>                      | 105     |
| <code>\markdownRendererTableAttributeContextBegin</code> | 104     |
| <code>\markdownRendererTableAttributeContextEnd</code>   | 104     |
| <code>\markdownRendererTextCite</code>                   | 76      |
| <code>\markdownRendererThematicBreak</code>              | 106     |
| <code>\markdownRendererTickedBox</code>                  | 107     |
| <code>\markdownRendererUlBegin</code>                    | 74      |
| <code>\markdownRendererUlBeginTight</code>               | 74      |
| <code>\markdownRendererUlEnd</code>                      | 75      |

|                                                |                                |
|------------------------------------------------|--------------------------------|
| <code>\markdownRendererUListTight</code>       | 75                             |
| <code>\markdownRendererUListItem</code>        | 74                             |
| <code>\markdownRendererUListItemEnd</code>     | 75                             |
| <code>\markdownRendererUntickedBox</code>      | 107                            |
| <code>\markdownSetup</code>                    | 52, 52, 56, 126, 132, 351, 352 |
| <code>\markdownSetupSnippet</code>             | 65, 65                         |
| <code>\markdownWarning</code>                  | 123, 123                       |
| <code>new</code>                               | 7, 16, 324                     |
| <code>notes</code>                             | 34, 94                         |
| <code>parsers</code>                           | 204, 237                       |
| <code>parsers.punctuation</code>               | 205                            |
| <code>pipeTables</code>                        | 6, 35, 41, 105                 |
| <code>preserveTabs</code>                      | 35, 39, 238                    |
| <code>rawAttribute</code>                      | 35, 36, 99                     |
| <code>reader</code>                            | 8, 26, 134, 204, 237, 279      |
| <code>reader-&gt;add_special_character</code>  | 8, 9, 26, 272                  |
| <code>reader-&gt;auto_link_email</code>        | 262                            |
| <code>reader-&gt;auto_link_url</code>          | 262                            |
| <code>reader-&gt;create_parser</code>          | 238                            |
| <code>reader-&gt;finalize_grammar</code>       | 268, 329                       |
| <code>reader-&gt;initialize_named_group</code> | 273                            |
| <code>reader-&gt;insert_pattern</code>         | 8, 9, 26, 268, 269, 274        |
| <code>reader-&gt;lookup_reference</code>       | 250                            |
| <code>reader-&gt;normalize_tag</code>          | 238                            |
| <code>reader-&gt;options</code>                | 237                            |
| <code>reader-&gt;parser_functions</code>       | 238                            |
| <code>reader-&gt;parser_functions.name</code>  | 238                            |
| <code>reader-&gt;parsers</code>                | 237, 237                       |
| <code>reader-&gt;register_link</code>          | 250                            |
| <code>reader-&gt;update_rule</code>            | 268, 271, 274                  |
| <code>reader-&gt;writer</code>                 | 237                            |
| <code>reader.new</code>                        | 237, 237, 329                  |
| <code>relativeReferences</code>                | 36                             |
| <code>\setupmarkdown</code>                    | 132                            |
| <code>shiftHeadings</code>                     | 6, 37                          |
| <code>singletonCache</code>                    | 16                             |
| <code>slice</code>                             | 6, 37, 186, 198                |
| <code>smartEllipses</code>                     | 38, 82, 127                    |
| <code>\startmarkdown</code>                    | 131, 132, 380                  |

|                               |                                         |
|-------------------------------|-----------------------------------------|
| startNumber                   | 38, 95–97                               |
| \stopmarkdown                 | 131, 132, 380                           |
| strikeThrough                 | 38, 103, 376                            |
| stripIndent                   | 39, 238                                 |
| stripPercentSigns             | 345                                     |
| subscripts                    | 39, 104                                 |
| superscripts                  | 40, 104                                 |
| syntax                        | 270, 274                                |
| tableAttributes               | 40, 104                                 |
| tableCaptions                 | 6, 40, 41, 104                          |
| taskLists                     | 41, 107, 366                            |
| texComments                   | 42, 239                                 |
| texMathDollars                | 42, 106                                 |
| texMathDoubleBackslash        | 43, 106                                 |
| texMathSingleBackslash        | 43, 106                                 |
| tightLists                    | 43, 74, 75, 80, 82, 94, 95, 98, 99, 357 |
| underscores                   | 44                                      |
| util.cache                    | 135, 135                                |
| util.cache_verbatim           | 135                                     |
| util.encode_json_string       | 136                                     |
| util.err                      | 135                                     |
| util.escaper                  | 138                                     |
| util.expand_tabs_in_line      | 136                                     |
| util.flatten                  | 137                                     |
| util.intersperse              | 138                                     |
| util.map                      | 138                                     |
| util.pathname                 | 139                                     |
| util.rope_last                | 137                                     |
| util.rope_to_string           | 137                                     |
| util.table_copy               | 136                                     |
| util.walk                     | 136, 137                                |
| walkable_syntax               | 8, 17, 23, 268, 269, 271–274            |
| writer                        | 134, 134, 186, 279                      |
| writer->active_attributes     | 197, 197, 198                           |
| writer->attribute_type_levels | 197                                     |
| writer->attributes            | 195                                     |
| writer->block_html_element    | 193                                     |
| writer->blockquote            | 194                                     |
| writer->bulletitem            | 192                                     |
| writer->bulletlist            | 191                                     |

|                                  |          |
|----------------------------------|----------|
| writer->citations                | 280      |
| writer->code                     | 190      |
| writer->contentblock             | 284      |
| writer->defer_call               | 203, 203 |
| writer->definitionlist           | 287      |
| writer->display_math             | 317      |
| writer->div_begin                | 299      |
| writer->div_end                  | 300      |
| writer->document                 | 194      |
| writer->ellipsis                 | 188      |
| writer->emphasis                 | 193      |
| writer->escape                   | 190      |
| writer->escape_minimal           | 189      |
| writer->escape_programmatic_text | 189      |
| writer->escape_typographic_text  | 189      |
| writer->escaped_chars            | 189, 189 |
| writer->escaped_minimal_strings  | 188, 189 |
| writer->escaped_strings          | 189      |
| writer->escaped_uri_chars        | 188, 189 |
| writer->fancyitem                | 290      |
| writer->fancylist                | 289      |
| writer->fencedCode               | 295      |
| writer->flatten_inlines          | 186, 186 |
| writer->get_state                | 203      |
| writer->hard_line_break          | 188      |
| writer->heading                  | 201      |
| writer->identifier               | 190      |
| writer->image                    | 191      |
| writer->infostring               | 190      |
| writer->inline_html_comment      | 193      |
| writer->inline_html_tag          | 193      |
| writer->inline_math              | 318      |
| writer->interblocksep            | 188      |
| writer->is_writing               | 186, 186 |
| writer->jekyllData               | 321      |
| writer->lineblock                | 305      |
| writer->link                     | 191      |
| writer->mark                     | 306      |
| writer->math                     | 190      |
| writer->nbsp                     | 187      |
| writer->note                     | 309      |
| writer->options                  | 186      |

|                         |               |
|-------------------------|---------------|
| writer->ordereditem     | 193           |
| writer->orderedlist     | 192           |
| writer->pack            | 187, 278      |
| writer->paragraph       | 187           |
| writer->paragraphsep    | 188           |
| writer->plain           | 187           |
| writer->pop_attributes  | 197, 198      |
| writer->push_attributes | 197, 197, 198 |
| writer->rawBlock        | 295           |
| writer->rawInline       | 315           |
| writer->set_state       | 203           |
| writer->slice_begin     | 186           |
| writer->slice_end       | 186           |
| writer->soft_line_break | 188           |
| writer->space           | 187           |
| writer->span            | 279           |
| writer->strike_through  | 315           |
| writer->string          | 190           |
| writer->strong          | 194           |
| writer->subscript       | 316           |
| writer->suffix          | 187           |
| writer->superscript     | 317           |
| writer->table           | 312           |
| writer->thematic_break  | 188           |
| writer->checkbox        | 194           |
| writer->undosep         | 188, 277      |
| writer->uri             | 190           |
| writer->verbatim        | 194           |
| writer.new              | 186, 186, 329 |